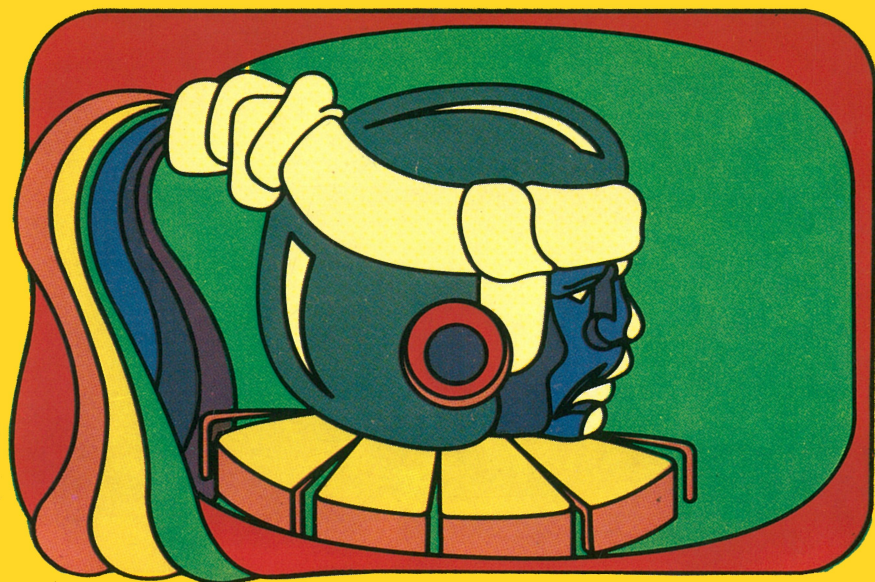
8080A/8085:

PROGRAMMAZIONE IN LINGUAGGIO ASSEMBLY

EDIZIONE ITALIANA

LANCE A. LEVENTHAL

GRUPPO EDITORIALE JACKSON

8080A/8085:

PROGRAMMAZIONE IN LINGUAGGIO ASSEMBLY

di
Lance
A. Leventhal

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

RINGRAZIAMENTI

L'Autore vuole esprimere la propria riconoscenza al:
Sig. William Tester del Grossmont College, che ha reso possibile questo libro; Sig. Colin Walsh del Grossmont College, che ha collaborato al disegno e sviluppo degli esempi di programmazione; Sig. Curt Ingraham di Osborne & Associates, che ha fatto numerose correzioni editoriali; Sig.a Teddy Ferguson, che ha composto gli assegnamenti del problema originario; Sig. Stanley Rogers della Society for Computer Simulation, che, con convincenti suggerimenti, ha portato molti miglioramenti allo stile di scrittura degli autori e sua moglie Donna per la sua costanza e comprensione per l'intera scrittura di questo libro.

Altre persone che hanno fornito assistenza e suggerimenti sono state: Sig. Jeffrey Haight, Prof. Nicholas Panos, Sig. David Bulman, Sig.a Kati Bulman, Sig. Bernard Laffreniere, Sig. Charles Robe, Sig. Michael Viehman, Sig. Richard Evans, Sig. Frederick Lepow e Sig. William Long. Altri studenti e colleghi hanno aiutato l'autore a mantenere la via corretta.

L'Autore, naturalmente, è direttamente responsabile per tutti gli errori rimasti, idee ed interpretazioni errate.

Questo libro è dedicato alla memoria di Bayar Goodman, il quale avrebbe apprezzato con gioia questa nuova tecnologia.

SOMMARIO

SOMMARIO (continua)

CAPITOLO **PAGINA**

SOMMARIO (continua)

SOMMARIO (continua)

SOMMARIO (continua)

SOMMARIO (continua)

SOMMARIO (continua)

SOMMARIO (continua)

CAPITOLO **PAGINA**

INDICE ANALITICO

INDICE ANALITICO (continua)

INDICE ANALITICO (continua)

INDICE ANALITICO (continua)

INDICE ANALITICO (continua)

INDICE ANALITICO (continua)

INDICE DELLE TABELLE

INDICE DELLE FIGURE

INDICE DELLE FIGURE (continua)

Capitolo 1

INTRODUZIONE AL LINGUAGGIO DI PROGRAMMAZIONE ASSEMBLY

Questo libro descrive la programmazione in linguaggio assembly. Si presuppone noto il testo An Introduction To Microcomputers: Volume I — Basic Concepts (particolarmente i Capitoli 6 e 7). Questo libro non comprende una discussione sulle caratteristiche generali dei computer, microcomputer, metodi di indirizzamento, o set di istruzioni; per tali informazioni si può far riferimento a An Introduction To Microcomputers: Volume I.

COME È STATO STAMPATO QUESTO LIBRO

Si noti che il testo di questo libro è stato stampato con molte parti in grassetto. Questo serve per saltare quelle parti con le quali si ha familiarità. In ogni caso la stampa in caratteri normali ha solo la funzione di espandere un'informazione precedentemente presentata in grassetto. Perciò si legga solo la parte in grassetto fino a quando non si trovi un argomento che si vuole approfondire, a questo punto si inizi la lettura delle parti in carattere ordinario.

IL SIGNIFICATO DELLE ISTRUZIONI

Il set di istruzioni di un microprocessore è semplicemente il set di ingressi binari che originano ben definite azioni durante un ciclo di istruzione. Un set di istruzione, nei confronti di un microprocessore, è come la tabella funzionale per un dispositivo logico come un gate, un sommatore, un registro di Scorrimento. Naturalmente le azioni che il microprocessore esegue in risposta agli ingressi di istruzione sono molto più complesse delle azioni che i dispositivi a logica combinatoria eseguono in risposta ai loro ingressi.

ISTRUZIONI BINARIE

Una istruzione è semplicemente una struttura di bit binari — essa deve essere presente agli ingressi dei dati del microprocessore ad un istante opportuno per essere interpretata come istruzione. Per esempio nel caso del microprocessore 8080 quando la struttura degli 8 bit binari 10000000 è l'ingresso durante l'operazione di esecuzione di istruzioni, significa:

«Somma il contenuto del registro B al contenuto dell'Accumulatore».

Analogamente la struttura 00111110 significa:

«Posiziona il contenuto della prossima parola della memoria di programma nell'Accumulatore».

Il microprocessore (come qualsiasi altro computer) ispeziona soltanto strutture binarie come istruzioni o dati. Esso non ispeziona parole o numeri ottali, decimali od esadecimali.

UN PROGRAMMA DEL COMPUTER

Un programma è una sequenza di istruzioni per mezzo delle quali un computer esegue un particolare compito.

PROGRAMMA DEL COMPUTER

Attualmente un programma del computer non contiene solo istruzioni; esso contiene anche i dati e gli indirizzi di memoria che il microprocessore richiede per eseguire il compito definito dalle istruzioni. Chiaramente, se il microprocessore deve eseguire un'addizione, esso deve avere due numeri da sommare ed una locazione per il risultato. Il programma del computer deve in qualche modo determinare la sorgente dei dati e la locazione del risultato oltre all'operazione da eseguire. Tutti i microprocessori eseguono sequenzialmente le istruzioni fino a quando una delle istruzioni non cambia la sequenza di esecuzione o arresta il computer. Cioè il microprocessore considera la prossima istruzione dal prossimo indirizzo di memoria consecutivo fino a quando non è comandato diversamente dalla istruzione attuale.
Alla fine tutto il programma diventa un set di numeri binari. Per esempio il programma per l'8080 che somma i contenuti delle locazioni di memoria 60_{16} e 61_{16} e pone il risultato nella locazione 62_{16} è il seguente:

```
00111010
01100000
00000000
01000111
00111010
01100001
00000000
10000000
00110010
01100010
00000000
```

PROGRAMMA OGGETTO

PROGRAMMA IN LINGUAGGIO DI MACCHINA

Questo è un linguaggio di macchina, o programma oggetto. Se questo programma viene mandato nella memoria di un microcomputer basato sull'8080, il microcomputer è in grado di seguirlo direttamente.

IL PROBLEMA DELLA PROGRAMMAZIONE

Esistono molte difficoltà associate con la creazione di programmi come programmi oggetto, o in linguaggio binario di macchina. Alcuni dei problemi sono:

1) I programmi sono difficili da capire o al debugging (correzione degli errori) (le strutture binarie sembrano tutte uguali, in modo particolare dopo averle guardate per alcune ore).
2) I programmi sono lenti ad entrare perchè deve entrare ogni bit singolarmente.
3) I programmi non descrivono il compito che si vuole fare eseguire al computer in un formato di lettura simile a quello umano.
4) I programmi sono lunghi e fastidiosi da scrivere.
5) Il programmatore spesso commette errori di distrazione che sono successivamente molto difficili da trovare.

Per esempio, **la seguente versione del programma oggetto per l'addizione presenta un solo errore di bit. Si cerchi di trovarlo:**

00111010
01100000
00000000
01000111
01110010
01100001
00000000
10000000
00110010
01100010
00000000

Mentre il calcolatore lavora con facilità sui numeri binari non è così per l'uomo. Infatti per l'uomo i programmi binari sono lunghi, fastidiosi, confusi, incomprensibili. Eventualmente un programmatore può iniziare ricordando alcuni codici binari, ma tale sforzo potrebbe essere impiegato molto più produttivamente.

L'USO DELL'OTTALE O DELL'ESADECIMALE

OTTALE O ESADECIMALE

Tavolta si può migliorare la situazione usando, piuttosto che i numeri binari, il sistema ottale o esadecimale. In questo libro si useranno i numeri esadecimali perchè, oltre ad essere più brevi, sono lo standard dell'industria del microprocessore. La Tabella 1-1 definisce i numeri base del sistema esadecimale e gli equivalenti binari. **Il programma 8080 per sommare due numeri ora diventa:**

3A
60
00
47
3A
61
00
80
32
62
00

Almeno la versione esadecimale è più breve da scrivere e non molto difficile da esaminare. **Talvolta gli errori sono più facili da trovare in una sequenza di numeri esadecimali. La versione errata del programma dell'addizione, in forma esadecimale diventa:**

3A
60
00
47
72
61
00
80
32
62
00

L'errore è più facile da rilevare.

Cosa si vuol fare col programma esadecimale? Il microprocessore comprende soltanto codici di istruzione binarie. L'ostacolo si supera convertendo i numeri esadecimali in numeri binari. Questa conversione è un compito ripetitivo e tedioso. Coloro che hanno tentato di farla commettono tutti i tipi di errori di distrazione, come considerazione della riga sbagliata, salto di un bit o trasposizione di un bit o di un numero.

CARICATORE ESADECIMALE

Comunque questo compito ripetitivo ed estenuante è un lavoro perfetto per un computer. Il computer è instancabile e non commette errori banali. **Quindi l'idea è scrivere il programma in numeri esadecimali e poi convertirli in numeri binari. Questo è uno standard di programma fornito con molti microprocessori; esso è chiamato «caricatore esadecimale».**
Conviene avere un caricatore esadecimale? Se si vuole scrivere un programma in numeri binari e se il programma può entrare in forma binaria nel computer chiaramente il caricatore esadecimale non serve.
Evidentemente il caricatore esadecimale ha un certo costo, è un programma in più da caricare in memoria e quindi occuperà y locazioni di memoria – memoria che poteva essere diversamente impiegata.
Perciò esiste un compromesso tra richiesta di memoria e costo da parte del caricatore esadecimale in funzione del guadagno in tempo del programmatore.
Un caricatore esadecimale vale bene il suo piccolo costo.

Tabella 1-1
Tabella Di Conversione Esadecimale

Digit Esadecimale	Binario Equivalente	Decimale Equivalente
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Un caricatore esadecimale certamente non risolve tutti i problemi della programmazione. La versione esadecimale del programma è ancora difficile da leggere o comprendere; per esempio, essa non fa distinzioni tra dati e indirizzi, come pure la lista del programma non fornisce alcun suggerimento su quello che il programma fa. Cosa significa 32 o 47 o 3A? Appare quindi interessante la memorizzazione di una scheda con i codici più difficili. Inoltre il codice sarà completamente diverso per un altro microprocessore ed il programma richiede un certo numero di documentazioni.

MNEMONICI DEI CODICI DI ISTRUZIONE

Un sostanziale miglioramento della programmazione consiste nell'assegnare un nome ad ogni codice di istruzione. Il nome del codice di istruzione è detto «mnemonico» perchè aiuta la memoria. Il mnemonico dell'istruzione potrebbe descrivere in qualche modo quello che fa l'istruzione.

PROBLEMA CON I MNEMONICI

Infatti tutti i microprocessori costruiti (si ricordi anche il codice esadecimale) forniscono un set di mnemonici per il set di istruzioni del microprocessore. **Però non è necessario prendere in assoluto i mnemonici del costruttore;** non sono niente di sacro. Comunque essi sono standard per un dato microprocessore e perciò comprensibili da tutti gli utilizzatori. I nomi delle istruzioni si possono trovare su manuali, schede, libri, articoli e programmi. Il problema della selezione dei mnemonici di istruzione è che non tutte le istruzioni hanno un nome «ovvio». Alcune istruzioni hanno nomi ovvi (es. ADD, AND, OR) altre hanno contrazioni ovvie (es. SUB per la sottrazione, XOR per l'OR esclusivo), mentre altre ancora non hanno nè l'uno nè l'altro. Ne risultano mnemonici come WMP, PCHL ed anche SOB (si provi ad azzardare cosa possono significare!) La maggior parte dei costruttori introduce alcuni nomi ragionevoli ed altri disperati. Comunque gli utilizzatori che inventano mnemonici propri raramente risultanto migliori del costruttore.
Assieme ai mnemonici di istruzione il costruttore normalmente assegna nomi ai registri della CPU. Come per i nomi di istruzione, alcuni nomi di registro sono ovvi (es. A per Accumulatore) mentre altri possono avere solo significato storico. Inoltre qui si useranno i suggerimenti del costruttore semplicemente per favorire la standardizzazione.

LINGUAGGIO DI PROGRAMMAZIONE ASSEMBLY

Se si usano i mnemonici standard di istruzione e di registro per l'8080, come definiti dalla Intel, il programma per l'addizione dell'8080 diventa:

```
LDA
60
00
MOV B,A
LDA
61
00
ADD B
STA
62
00
```

Il programma è ancora lontano dall'essere ovvio ma almeno alcune parti sono comprensibili. ADD B è un considerevole miglioramento rispetto ad 80; LDA, MOV e STA possono suggerire caricamento, movimento e memorizzazione rispettivamente. Ora si deve conoscere quali righe sono di istruzione e quali di dati o indirizzi. **Questo è un programma redatto in linguaggio assembly.**

L'ASSEMBLATORE DEL PROGRAMMA

ASSEMBLY MANUALE

Come si può inserire il programma in linguaggio assembly nel computer? È necessario trasformarlo in numeri esadecimali o binari. **Si può trasformare manualmente,** istruzione per istruzione, **un programma in linguaggio assembly.** Questo è chiamato assembly manuale.
L'assembly manuale di una sequenza di tre istruzioni può essere illustrata come segue:

Nome dell'Istruzione	Esadecimale Equivalente
ADD B	80
LDA	3A
STA	32

Come nel caso della conversione da esadecimale a binario, l'assembly manuale è un compito meccanico quindi non interessante, ripetitivo e soggetto a numerosi errori secondari. Scelta della riga sbagliata, trasposizione di caratteri, omissione di istruzioni e lettura non corretta del codice sono solo alcuni degli errori che si possono compiere. La maggior parte dei microprocessori complicano ulteriormente il compito poichè presenta istruzioni a diversa lunghezza di parola. Alcune istruzioni sono composte da una parola mentre altre da due o tre. Alcune istruzioni richiedono dati nella seconda e terza parola, altri richiedono indirizzi di memoria, numeri di registro, ecc..

ASSEMBLATORE

PROGRAMMA SORGENTE

PROGRAMMA OGGETTO

L'assembly è un altro compito meccanico che si può assegnare al microcomputer. Il microcomputer non commette errori nella conversione di codici; esso conosce sempre quante parole e quale formato è richiesto da ogni istruzione. Il programma che esegue questo compito è detto «assemblatore». Il programma assemblatore trasforma un programma utente, o programma «sorgente» scritto con mnemonici, in un programma in linguaggio di macchina, o programma «oggetto», comprensibile al microcomputer. L'ingresso dell'assemblatore è un programma sorgente e la sua uscita è un programma oggetto.

Il compromesso discusso a proposito del caricatore esadecimale è ingrandito nel caso dell'assemblatore. Gli assemblatori sono molto più costosi, occupano più memoria, e richiedono più periferiche e tempo di esecuzione dei caricatori esadecimali. Mentre gli utenti possono (e spesso lo fanno) sostituirsi ai propri caricatori, più difficile è per gli assemblatori.

Gli Assemblatori hanno le proprie regole che si deve imparare a rispettare. Questo comprende l'uso di certi caratteri speciali (come spazi, virgole, due punti o punti) in posizione appropriata, scansione corretta, il controllo conveniente dell'informazione ed anche il corretto posizionamento di nomi e numeri. Queste regole sono tipicamente il minor ostacolo, facilmente superabile.

CARATTERISTICHE ADDIZIONALI DEGLI ASSEMBLATORI

I primi programmi assemblatori non fanno niente di più che trasformare i nomi mnemonici di istruzioni e registri nei loro equivalenti binari. Comunque la maggior parte degli assemblatori ora fornisce caratteristiche addizionali come:

1) Consentire all'utente l'assegnazione di nomi alle locazioni di memoria, dispositivi di ingresso ed uscita ed anche sequenze di istruzioni.
2) Conversione di dati o indirizzi da vari sistemi numerici (es. decimale o esadecimale) al binario e caratteri di conversione nei loro codici binari ASCII o EBCDIC.
3) Esecuzione di alcune operazioni aritmetiche come parte del processo assembly.
4) Specificazione al caricatore di programma in quali parti di memoria può caricare il programma o i dati.
5) Possibilità da parte dell'utente di assegnare aree di memoria per l'immagazzinamento temporaneo di dati e per posizionare dati fissi in aree della memoria di programma.
6) Rifornimento delle informazioni richieste per acquisire programmi standard da biblioteche, o programmi scritti in tempi diversi ed inserimento nel programma attuale.
7) Possibilità da parte dell'utente di controllare il formato della lista del programma ed i dispositivi di ingresso ed uscita impiegati.

SCELTA DELLO ASSEMBLATORE

Tutte queste caratteristiche, naturalmente, comportano ulteriore costo ed impiego di memoria. I microcomputer generalmente hanno assemblatori molto più semplici rispetto ai grossi computer, però c'è sempre la tendenza all'aumento delle dimensioni dell'assemblatore. Spesso è necessario fare una scelta di assemblatori. Un criterio importante per tale scelta non è il numero delle caratteristiche ausiliarie dell'assemblatore, ma piuttosto la convenienza del lavoro praticato normalmente.

SVANTAGGI DEL LINGUAGGIO ASSEMBLY

L'assemblatore, analogamente al caricatore esadecimale, non risolve tutti i problemi della programmazione. Un problema è l'enorme vuoto tra il set di istruzione del microcomputer ed i compiti che il microcomputer deve eseguire. Le istruzioni del computer tendono a fare cose come sommare il contenuto di due registri, spostare il contenuto dell'Accumulatore di un bit, o sostituire un nuovo valore nel Contatore di Programma. D'altra parte l'utente generalmente richiede al microcomputer di valutare, per esempio, se una lettura analogica ha superato una soglia, controllare e comandare un particolare comando da una telescrivente, oppure attivare un relè all'istante opportuno. Un programmatore in linguaggio assembly deve tradurre tali richieste in una sequenza di semplici istruzioni del computer. Tale traduzione può essere un lavoro lungo e difficile.
Inoltre, **se si sta programmando in linguaggio assembly è necessaria una dettagliata conoscenza del particolare microcomputer che si sta usando.** Occorre conoscere con precisione i registri e le istruzioni di cui è dotato il microcomputer, in particolare come le istruzioni influenzano i vari registri, quali metodi di indirizzamento impiega il computer ed una miriade di altre informazioni. Nessuna di queste informazioni è importante rispetto al compito che il microcomputer deve alla fine eseguire.

PORTABILITÀ

Inoltre i programmi in linguaggio assembly non sono portabili. Ogni microcomputer ha il proprio linguaggio assembly, che riflette la propria architettura. Un programma in linguaggio assembly scritto per l'8080 non funzionerà sul Motorola 6800, sul Fairchild F8, o sul National Semiconductor PACE. Per esempio il programma per l'addizione scritto per il Motorola 6800 sarebbe:

```
LDAA    $60
ADDA    $61
STAA    $62
```

La mancanza di portabilità non solo impedisce l'uso del programma in linguaggio assembly in un altro microcomputer, ma impedisce anche l'uso di qualsiasi programma non specificatamente scritto per il microcomputer che si sta usando. Questo è un particolare inconveniente per i microcomputer dato che per questi dispositivi esistono pochi e nuovi programmi in linguaggio assembly. Il risultato è, troppo spesso, di trovarsi senza assistenza. Se si cerca un programma da eseguire in un particolare problema, è probabile che non si riesca a trovarlo in una piccola biblioteca di programmi che il costruttore fornisce. Probabilmente non si riuscirà a trovarlo in un archivio, in articoli o nello schedario di qualche vecchio programma. Probabilmente sarà necessario scriverlo da soli.

LINGUAGGI AD ALTO LIVELLO

COMPILATORE

La soluzione di molte delle difficoltà associate con i programmi in linguaggio assembly è quella di usare, al loro posto, linguaggi ad «alto livello» o «orientati alla procedura». Tali linguaggi consentono di descrivere compiti nella forma orientata al problema piuttosto che orientata al computer. Ogni statement in un linguaggio ad alto livello esegue una precisa funzione; questa corrisponderà generalmente a molte istruzioni in linguaggio assembly. Un programma chiamato compilatore trasforma il programma sorgente in linguaggio ad alto livello a programma oggetto od istruzioni in linguaggio di macchina.

FORTRAN

A seconda delle esigenze esistono molti linguaggi ad alto livello diversi. Se, per esempio, si vuole imporre al computer di eseguire una notazione algebrica, si può scrivere il programma in FORTRAN (Linguaggio per Traduzione di Formula), il vecchissimo e più largamente usato dei linguaggi ad alto livello. Ora, se si vuole sommare due numeri, basta dire al computer:

```
SUM = NUMB 1 + NUMB 2
```

Questo è molto più semplice (e molto più breve) del programma equivalente in linguaggio di macchina o dell'equivalente programma in linguaggio assembly. Altri linguaggi ad alto livello sono COBOL (per applicazioni commerciali), ALGOL (un altro linguaggio algebrico), PL/I (una combinazione di FORTRAN, ALGOL e COBOL) ed APL e BASIC (linguaggi comuni per sistemi a divisione di tempo o time-sharing).

VANTAGGI DEI LINGUAGGI AD ALTO LIVELLO

Chiaramente i linguaggi ad alto livello rendono più facile e veloce la scrittura dei programmi. Si stima comunemente che un programmatore può scrivere un programma in linguaggio ad alto livello circa dieci volte più velocemente rispetto al linguaggio assembly. Questo soltanto per la scrittura del programma, senza considerare la definizione del problema, la struttura del programma, debugging, collaudo, o documentazione, i quali diventano tutti più semplici e più veloci. Il programma in linguaggio ad alto livello, per esempio, è parzialmente auto-documentante. Anche se non si conosce il FORTRAN probabilmente si comprende cosa fa lo statement precedentemente illustrato.

INDIPENDENZA DALLA MACCHINA DEI PROGRAMMI AD ALTO LIVELLO

I linguaggi ad alto livello risolvono molti altri problemi associati con la programmazione in linguaggio assembly. Il linguaggio ad alto livello ha una propria sintassi (normalmente definita da uno stàndard nazionale od internazionale). Il linguaggio non dipende dal set di istruzioni, registri, od altre caratteristiche di un particolare computer. Il compilatore si fa carico di tutti questi dettagli. I programmatori possono concentrarsi sui loro problemi, non è richiesta una dettagliata comprensione dell'architettura della CPU che si vuole usare, per questo scopo non è richiesta nessuna conoscenza sul particolare computer che stanno programmando.

PORTABILITÀ DEI LINGUAGGI AD ALTO LIVELLO

I programmi scritti in linguaggio ad alto livello sono portabili — almeno in teoria. Essi possono essere eseguiti in qualsiasi computer o microcomputer che possiede un compilatore standard per quel linguaggio. Analogamente tutti i precedenti programmi scritti in un linguaggio ad alto livello per precedenti computer sono disponibili quando si programma un nuovo computer. Questo può significare centinaia di programmi nel caso di linguaggi comuni come FORTRAN o BASIC.

SVANTAGGI DEI LINGUAGGI AD ALTO LIVELLO

Se tutti i vantaggi elencati per i programmi ad alto livello sono veri, se si può scrivere programmi più velocemente e renderli inoltre portabili, perché il fastidio dei linguaggi assembly? Vogliamo tormentarci con registri, codici di istruzione, mnemonici, e tutte le altre seccature! Naturalmente ci sono degli svantaggi che bilanciano i vantaggi.

SINTASSI DEI LINGUAGGI AD ALTO LIVELLO

Un problema ovvio **è la necessità di imparare le «regole» o «sintassi» di qualsiasi linguaggio ad alto livello** che si vuole usare. Un linguaggio ad alto livello ha un set di regole abbastanza complicato. Si vedrà che molto tempo occorrerà dedicarlo alla cura affinchè il programma sia sintatticamente corretto (anche se poi probabilmente il programma non farà quello che si voleva). Un linguaggio ad alto livello del computer è come un linguaggio estero. Con un po' di talento si possono usare le regole per ricavare i programmi che il compilatore accetterà. Inoltre imparando le regole e cercando di fare accettare il programma al compilatore non contribuisce al compito che si vuole risolvere.

Così, per esempio, alcune regole del FORTRAN sono:

- Le label devono essere numerate nelle prime cinque colonne della scheda.
- Gli statement devono partire dalla colonna sette.
- Le variabili intere cominciano con le lettere I, J, K, L, M o N.

Un altro problema ovvio è dato **dalla necessità di un compilatore per trasformare i programmi scritti in linguaggio ad alto livello.** I compilatori sono costosi ed occupano una grande quantità di memoria. Mentre la maggior parte degli assemblatori occupa da 2K a 16K byte di memoria (1K = 1024), i compilatori occupano da 4K a 64K byte. Tale dispendio è giustificato solo se il compilatore viene molto usato.

COSTO DEI COMPILATORI

Inoltre **solo pochi compilatori faranno sì che la rappresentazione del problema sia più semplice.** Il FORTRAN, per esempio, è particolarmente adatto per problemi che possono essere espressi in formule algebriche. Comunque se il problema è il controllo di una stampante, la redazione di una stringa di caratteri o l'osservazione di un sistema di allarme è difficile esprimerlo in notazione algebrica. Infatti la formulazione della soluzione in notazione algebrica può essere più goffa e difficoltosa della formulazione in linguaggio assembly. La risposta è di usare, naturalmente, un linguaggio ad alto livello più adatto. Esistono alcuni linguaggi di questo tipo ma essi sono meno largamente usati e standardizzati del FORTRAN. Non si avranno molti dei vantaggi dei linguaggi ad alto livello se si usano i linguaggi cosiddetti ad implementazione di sistema.

NOTAZIONE ALGEBRICA

I linguaggi ad alto livello originano programmi in linguaggio di macchina non molto efficienti. Il motivo principale di questo fatto è che la compilazione è un processo automatico che viene vagliato con compromessi per permettere molti range e possibilità. Il compilatore lavora principalmente come un trasformatore di linguaggi computerizzati — talvolta le parole sono corrette ma i suoni e le strutture risultanti sono inopportune. Un semplice compilatore non può conoscere quando una variabile non sarà più usata e quindi può essere scaricata, o quando un registro potrebbe essere usato come locazione di memoria o quando tra variabili esistono semplici relazioni. Il programmatore esperto può trarre vantaggio da riduzioni del tempo di esecuzione o dell'impiego di memoria. Solo pochi compilatori (noti come compilatori ottimizzati) possono fare anche questo, però come compilatori sono molto più grandi e lenti dei normali compilatori.

INEFFICENZA DEI PROGRAMMI AD ALTO LIVELLO

In generale i vantaggi e gli svantaggi dei linguaggi ad alto livello sono:

Vantaggi:

VANTAGGI DEI LINGUAGGI AD ALTO LIVELLO

- Descrizione più conveniente del problema.
- Più efficiente scrittura del programma.
- Più facile documentazione.
- Sintesi standard.
- Indipendenza della struttura del particolare computer.
- Portabilità.
- Disponibilità di biblioteca ed altri programmi.

Svantaggi:

SVANTAGGI DEI PROGRAMMI AD ALTO LIVELLO

- Regole speciali.
- Richiesta di hardware estensivo e di software di supporto.
- Orientazione di linguaggi comuni a problemi algebrici o commerciali.
- Programmi inefficienti.
- Difficoltà del codice ottimizzato a soddisfare richieste di tempo e memoria.
- Impossibilità di usare caratteristiche speciali del computer.

LINGUAGGI AD ALTO LIVELLO PER MICROPROCESSORI

Gli utenti del microprocessore incontreranno delle difficoltà particolari nell'impiego di linguaggi ad alto livello. Tra queste risultano:

- **Per i microprocessori esistono pochi linguaggi ad alto livello.**
- **Non sono facilmente disponibili linguaggi standard.**
- **Attualmente pochi compilatori servono i microcomputer. Quelli che lo fanno spesso richiedono una grande quantità di memoria.**
- **Molte applicazioni del microprocessore non sono adatte per linguaggi ad alto livello.**
- **I costi di memoria sono spesso critici nelle applicazioni del microprocessore.**

L'insufficienza di linguaggi ad alto livello è parzialmente dovuta al fatto che i microprocessori sono relativamente nuovi e sono i prodotti di costruttori di componenti a semiconduttore piuttosto che costruttori di computer.

Per i microprocessori esistono pochissimi linguaggi ad alto livello. I più comuni sono i linguaggi di tipo PL/I come il PL/M della Intel, l'MPL della Motorola ed il PLμS della Signetics.
Pur essendo pochi i linguaggi ad alto livello esistenti non consentono di individuare uno standard per cui l'utente del microprocessore non può aspettarsi di ottenere molta portatilità di programma, accesso alle biblioteche di programma, od uso di precedente esperienza o programmi. Il principale vantaggio deriva dalla riduzione dello sforzo di programmazione e dalla minore quantità di dettagli necessari per la comprensione dell'architettura del computer.

DISPENDIO PER I LINGUAGGI AD ALTO LIVELLO

L'uso di linguaggi ad alto livello per microprocessori è considerevolmente dispendioso. I microprocessori stessi sono più adatti per controllare e rallentare applicazioni interattive piuttosto che avere le caratteristiche di manipolazione ed analisi del linguaggio richieste nella compilazione. Perciò molti compilatori per microprocessori non servono in un sistema basato su microprocessore. Invece essi richiedono un computer molto più grande, cioè essi sono cross-compilatori piuttosto che autocompilanti. Quindi un utente non deve solo sostenere le spese di un computer più grosso ma deve anche trasferire fisicamente il programma da un computer più grande ad un micro.

Sono disponibili pochi auto-compilatori. Questi compilatori servono il microcomputer producendo il codice oggetto. Sfortunatamente essi richiedono grandi quantità di memoria (16K od oltre) più speciali hardware e software di supporto.

INOPPORTUNITÀ DEI LINGUAGGI ALTO LIVELLO

Generalmente i linguaggi ad alto livello non sono adatti per le applicazioni del microprocessore. La maggior parte dei linguaggi comuni sono stati escogitati sia per aiutare nella soluzione di problemi scientifici o per trattare la elaborazione su larga scala di dati commerciali. Poche applicazioni del microprocessore rientrano in entrambe queste aree. La maggior parte delle applicazioni del microprocessore comprendono dati di indirizzamento e controllo dell'informazione ai dispositivi di uscita e ricezione di dati con informazione dello stato dal dispositivo di ingresso. Spesso l'informazione di controllo e stato consiste in alcuni numeri binari con il significato di una ben precisa corrispondenza di hardware. Se si cerca un tipico programma di controllo in linguaggio ad alto livello ci si troverà molto a disagio. Per compiti in settori come forniture di test, terminali, sistemi di navigazione o forniture commerciali, i linguaggi ad alto livello lavorano molto meglio di quello che fanno nei settori strumentazione, comunicazioni, periferiche ed applicazioni di automazione.

AREE DI APPLICAZIONE PER LIVELLI DI LINGUAGGIO

Le applicazioni più adatte per linguaggi ad alto livello sono quelle che richiedono grandi memorie. Se il costo di un singolo chip di memoria è importante, come in un controllore di valvola, gioco elettronico, dispositivo controllore o piccolo strumento, allora l'inefficienza del linguaggio ad alto livello è intollerabile. D'altra parte se il sistema ha molte mi-

gliaia di byte di memoria per ogni via, come in un terminale di una fornitura di test, la inefficienza dei programmi ad alto livello non è importante. Chiaramente la dimensione del programma ed il volume del prodotto sono fattori molto importanti. Un grosso programma aumenterà enormemente i vantaggi dei linguaggi ad alto livello. D'altra parte l'applicazione di grandi volumi significherà un costo fisso di software di sviluppo che non è importante come i costi di memoria che sono una parte di ogni sistema.

QUALE LIVELLO SI DOVREBBE USARE?

Questo dipende dalla particolare applicazione. Si segnaleranno alcuni dei fattori che possono favorire particolari livelli:

Linguaggio di Macchina:

APPLICAZIONI PER IL LINGUAGGIO DI MACCHINA

- **Applicazioni a basso volume comprendenti piccoli, semplici programmi.**
- **Applicazioni dove il prototipo è il prodotto finale.**
- **Applicazioni di semplice controllo comprendenti un numero limitato di calcoli.**

Linguaggio Assembly:

APPLICAZIONI PER IL LINGUAGGIO ASSEMBLY

- **Programmi di dimensioni piccole e medie.**
- **Applicazioni dove è importante il costo della memoria.**
- **Applicazioni di controllo in tempo reale.**
- **Elaborazione limitata di dati.**
- **Applicazioni ad alto volume.**

Linguaggi ad Alto Livello:

APPLICAZIONI PER IL LINGUAGGIO AD ALTO LIVELLO

- **Grossi programmi.**
- **Applicazioni a basso volume richiedenti lunghi programmi.**
- **Applicazioni richiedenti grandi memorie.**
- **Più calcoli che operazioni di ingresso/uscita o controllo.**
- **Compatibilità con applicazioni simili impieganti grossi computer.**
- **Disponibilità di specifici programmi, in un linguaggio ad alto livello, che possono essere usati nell'applicazione.**

Anche molti altri fattori sono importanti, come la disponibilità di un computer più grosso per lo sviluppo, l'esperienza con particolare linguaggi e compatibilità con altre applicazioni.
Se l'hardware è la parte più costosa dell'applicazione, si potrebbe favorire il linguaggio assembly. Però occorre essere preparati ad impiegare tempo ulteriore per lo sviluppo del software avendo guadagnato con minor costo di memoria e più alta velocità di esecuzione. Se il software sarà la parte più costosa dell'applicazione è conveniente un linguaggio ad alto livello. Ci saranno però altre spese per il supporto dell'hardware e del software.
Naturalmente è possibile usare entrambi i linguaggi assembly e ad alto livello. Inizialmente si può scrivere il programma in linguaggio ad alto livello e poi trasformare alcune parti in linguaggio assembly. Comunque la maggior parte degli utenti non fa questo perchè ha effetti disastrosi nel debugging, testing e documentazione.

PROSPETTIVE FUTURE

È verosimile pensare che il futuro tenderà a favorire i linguaggi ad alto livello per le seguenti ragioni:

TENDENZA DEL FUTURO DEI LIVELLI DI LINGUAGGIO

- I programmi tendono ad aggiungere sempre nuove caratteristiche ed a crescere sempre più.
- L'hardware e la memoria stanno diventando meno costosi.
- Il software ed i programmatori stanno diventando più dispendiosi.
- I chip di memoria stanno divenendo disponibili in dimensioni sempre maggiori, a più basso costo «per bit», cosicché l'attuale economia in chip appare meno probabile.
- Sono disponibili più compilatori.
- Si stanno sviluppando linguaggi ad alto livello più adatti e più efficienti.
- Sarà operata una maggiore standardizzazione dei linguaggi ad alto livello.

La programmazione in linguaggio assembly dei microprocessori supererà quello che è attualmente per i grandi computer. Però programmi più lunghi, memorie meno costose, e programmatori più dispendiosi attribuiranno al software una grossa parte del costo di molte applicazioni. Lo scoglio in molte applicazioni sarà perciò l'introduzione di linguaggi ad alto livello.

PERCHÈ QUESTO LIBRO?

Se il futuro sembra favorire i linguaggi ad alto livello, come si giustifica un libro sul linguaggio di programmazione assembly? Le ragioni sono:

1) La maggior parte degli utenti di microcomputer programma in linguaggio assembly (almeno 2/3, secondo una recente indagine).
2) Molti utenti di microcomputer continueranno a programmare in linguaggio assembly poichè hanno bisogno del dettagliato controllo che questo fornisce.
3) Non è ancora stato prodotto un linguaggio ad alto livello destinato a diventare largamente disponibile o standardizzato.
4) Molte applicazioni richiedono l'efficienza del linguaggio assembly.
5) La comprensione del linguaggio assembly può aiutare a valutare i linguaggi ad alto livello.

D'ora in poi si tratterà esclusivamente con assemblatori e linguaggio di programmazione assembly. Comunque si vuole far sapere ai lettori che il linguaggio assembly non è la sola alternativa. Si potrebbe osservare per nuovi sviluppi che esso può ridurre in modo significativo il costo della programmazione se tale costo è il maggior fattore nell'applicazione che si considera.

Capitolo 2
ASSEMBLATORI

Questo capitolo discute le funzioni eseguite dagli assemblatori, cominciando con le caratteristiche comuni alla maggior parte degli assemblatori prosegue poi verso capacità più sofisticate come le macro e l'assembly condizionale. È possibile non approfondire questo capitolo in prima istanza, ritornandovi in seguito risulterà più interessante.

CARATTERISTICHE DEGLI ASSEMBLATORI

Come già ricordato precedentemente, gli assemblatori attuali fanno molto di più che tradurre i mnemonici del linguaggio assembly in codice binario. Prima si descriverà quindi come un assemblatore opera la trasformazione dei mnemonici e poi seguirà la descrizione delle caratteristiche addizionali. Infine si spiegherà come gli assemblatori sono impiegati.

ISTRUZIONI DELL'ASSEMBLATORE

CAMPI DEL LINGUAGGIO ASSEMBLY

Le istruzioni del linguaggio assembly (dette anche «statement») sono divise in un certo numero di campi come mostrato nella Tabella 2-1. Il campo del codice di operazione è l'unico che non può mai essere vuoto; esso contiene sempre o un mnemonico di istruzione od un ordine all'assemblatore, chiamato pseudo-istruzione, pseudo-operazione, o pseudo-op.

Il campo di indirizzo può contenere un indirizzo o dati, o può essere bianco.

Tabella 2-1
I Campi Di Una Istruzione In Linguaggio Assembly

Campo della Label	Campo del Codice dell'Operazione o Mnemonico	Campo dell'operando o Indirizzo	Campo del Commento
START	LDA	VAL1	CARICA IL PRIMO NUMERO IN A
	MOV	B,A	SPOSTALO IN B
	LDA	VAL2	CARICA IL SECONDO NUMERO IN A
	ADD	B	SOMMA IL PRIMO NUMERO AD A
	STA	SUM	MEMORIZZA LA SOMMA
NEXT	?	?	PROSSIMA ISTRUZIONE
VAL1	DS		
VAL2	DS		
SUM	DS		

I campi del commento e della label sono opzionali. Un programmatore assegnerà una label ad uno statement o aggiungerà un commento a seconda della convenienza personale, per esempio, per rendere il programma di facile lettura.

FORMATO

Naturalmente l'assemblatore deve avere il modo per capire dove un campo finisce e ne inizia un altro. Gli assemblatori che usano schede perforate come ingresso spesso richiedono che ogni campo inizi in una precisa colonna della scheda. Questo è il formato fisso. Comunque i formati fissi possono essere non convenienti quando il mezzo di ingresso è il nastro di carta; i formati fissi sono fastidiosi anche per i programmatori. L'alternativa è avere un formato libero dove i campi possono comparire dovunque nella riga.

DELIMITATORI

Se un assemblatore non può usare la locazione della riga per dividere i campi, deve usare qualcos'altro. **La maggior parte degli assemblatori usa un simbolo speciale per delimitare l'inizio di ogni campo.** Il carattere più ovvio è lo spazio bianco. Le virgole, i punti, punto e virgola, due punti, slash, punti interrogativi ed altri caratteri che altrimenti non sarebbero usati nel linguaggio assembly, possono anche servire come delimitatori. La Tabella 2-2 elenca i delimitatori dell'assemblatore standard per l'Intel 8080.

Tabella 2-2
Delimitatori Standard Per L'Assemblatore Dell'8080

:	dopo una label
'spazio'	tra il codice dell'operazione e indirizzo
,	tra operandi nel campo indirizzi
;	prima di un commento

Ci si può esercitare un po' sui delimitatori. Alcuni assemblatori sono meticolosi riguardo ad ulteriori spazi o la comparsa di delimitatori in commenti o label. Un assemblatore ben scritto eliminerà questi problemi secondari, ma molti assemblatori non sono ben scritti. La nostra raccomandazione è semplice: se possibile si evitino potenzialmente i problemi.

Le seguenti regole possono essere di aiuto:

1) Non usare spazi ulteriori, particolarmente dopo virgole che separano operandi.
2) Non usare caratteri delimitatori in nomi o label.
3) Usare delimitatori standard anche se l'assemblatore non lo richiede. Tali programmi saranno poi assemblati da qualunque assembler.

LABEL

CAMPO DELLA LABEL

Il campo dalla label è il primo campo in una istruzione in linguaggio assembly; può essere lo spazio bianco se la label è presente, l'assemblatore assegna alla label il valore dell'indirizzo della locazione di memoria nella quale è caricato il primo byte del programma oggetto. Si può successivamente usare la label come un indirizzo o come dato nel campo di un altro operando di istruzione. L'assemblatore sostituirà la label con il valore assegnato in fase di creazione del programma oggetto.

LE LABEL NELLE ISTRUZIONI DI SALTO

Le label sono più frequentemente usate nelle istruzioni di Salto, Chiamata o Ramificazione. Queste istruzioni sostituiscono un nuovo valore nel Contatore di Programma e così alterano la normale esecuzione sequenziale delle istruzioni. JUMP 150_{16} significa «sostituisci il valore 150_{16} nel Contatore di Programma». La prossima istruzione eseguita sarà quella della locazione di memoria 150_{16}. L'istruzione JUMP START significa «posiziona il valore assegnato dalla label START nel Contatore di Programma». L'istruzione successivamente eseguita sarà quella a cui è stata assegnata la label START. La Tabella 2-3 contiene un esempio.

Tabella 2-3
Assegnazione Ed Impiego Di Una LABEL

```
          PROGRAMMA IN LINGUAGGIO ASSEMBLY
START        CARICA ACCUMULATORE 100
             •
             •
             • PROGRAMMA PRINCIPALE
             •
             •
             •
             JUMP START
```

Quando viene eseguita la versione in linguaggio di macchina di questo programma, la istruzione JUMP START fa in modo che l'indirizzo dell'istruzione con la label START venga posizionato nel Contatore di Programma. L'istruzione con la label START sarà quindi la prossima ad essere eseguita.

Perchè usare una label? Esistono diverse ragioni:

1) Una label rende più facile trovare e ricordare una locazione di memoria.
2) Le label possono essere variate per le correzioni del programma. Non è necessario cambiare nessuna label successiva di riferimento; l'assemblatore eseguirà tutte le variazioni necessarie.
3) L'assemblatore od il caricatore possono riallocare l'intero programma aggiungendo una costante (costante di riallocazione) ad ogni indirizzo nel quale è stata usata una label. Così può spostare il programma per permettere l'inserzione di altri programmi oppure un riordinamento della memoria.
4) Il programma è più facile da usare come programma di biblioteca, cioè è più facile per altri prendere un programma ed aggiungerlo ad alcuni programmi completamente diversi.
5) Non si ha un'idea della figura di uscita degli indirizzi di memoria. L'immagine di uscita degli indirizzi di memoria è particolarmente difficile con microprocessori che hanno istruzioni che variano in lunghezza.

Può meravigliare l'assegnazione di una label a tutte le istruzioni che si potrebbero usare come destinazione o, altrimenti, che si vuole identificare.

SCELTA DELLE LABEL

Ci si può chiedere quale label si deve usare. Gli assemblatori spesso impongono alcune restrizioni sul numero di caratteri (normalmente 5 o 6), il carattere di testa (spesso deve essere una lettera) ed i caratteri di coda (spesso devono essere lettere, numeri, od uno dei pochi caratteri speciali). A parte queste restrizioni la scelta dipende dal programmatore.
Per esperienza si consiglia di usare label che suggeriscano il loro scopo, cioè label mnemoniche. Esempi tipici sono ADDW in una routine che aggiunge una parola in una somma, SRETX in una routine che ricerca il carattere ETX del codice ASCII ovvero NKEYS per una locazione della memoria dati che contiene il numero degli ingressi del tasto. Label significative sono più facili da ricordare e contribuiscono alla documentazione del programma. Alcuni programmatori preferiscono usare un formato di label standard, come la partenza con L0000. Queste label sono autosequenzianti (si può saltare di alcuni numeri per permettere inserzioni), ma esse non possono aiutare la documentazione del programma.

REGOLE PER L'USO DELLE LABEL

Alcune regole di selezione della label elimineranno delle preoccupazioni. Si raccomandano le seguenti:

1) Non usare label uguali ai codici di operazione o ad altri mnemonici. La maggior parte degli assemblatori non consentirà questo impiego.
2) Non usare label più lunghe di quanto consentito dall'assemblatore. Gli assemblatori sono dotati di diverse regole di troncamento.
3) Evitare caratteri speciali (non algebrici e non numerici). Alcuni assemblatori non lo permettono altri ne permettono solo alcuni. La pratica più semplice è di legarsi a lettere e numeri.
4) Si inizi ogni label con una lettera. Label cosi fatte sono sempre accettabili.
5) Non si usino label che possono generare confusione con qualsiasi altra. Si eviti la lettera I, O e Z ed i numeri 0, 1 e 2. Si evitino anche cose come XXXX e XXXXX. Non c'è ragione di tentare di opporsi alla legge di Murphy. («Se una cosa può andare storta, andrà sicuramente storta»).
6) Quando non si è sicuri se una label è corretta, non usarla. Non si trarrà alcun reale vantaggio nello scoprire se l'assemblatore l'accetta.

Queste sono raccomandazioni, non regole. Non è obbligatorio seguirle ma è buona norma farlo per evitare molti problemi.

CODICI DI OPERAZIONE DELL'ASSEMBLATORE (MNEMONICI)

Il compito fondamentale di un assemblatore è la trasformazione di codici di operazione mnemonici nei loro equivalenti binari. L'assemblatore esegue questo compito usando una tabella fissa come si farebbe costruendo manualmente l'assembly.
L'assemblatore deve comunque fare la trasformazione più corretta dei codici di operazione. Esso **deve anche determinare in qualche modo quanti operandi di istruzione sono richiesti e di che tipo sono.** Questo può essere piuttosto complesso – alcune istruzioni (come un Halt) non hanno operandi, altre (come l'istruzione di una Addizione o di un Salto) ne hanno uno, mentre altre ancora (come un trasferimento tra registri od uno spostamento di bit multiplo) ne richiedono due. Alcune istruzioni possono anche ammettere alternative, per es., alcuni computer hanno istruzioni (come Spostamento o Cancellazione) che possono essere applicati all'accumulatore o ad una locazione di memoria. Non si discuterà come l'assemblatore opera queste distinzioni, si noterà soltanto che esso deve farle.

PSEUDO-OPERAZIONI

ORDINI ALL'ASSEMBLATORE

Alcune istruzioni del linguaggio assembly non sono direttamente trasformabili in istruzioni del linguaggio di macchina. Queste istruzioni sono ordini all'assemblatore; esso assegna il programma a certe aree di memoria, definisce i simboli, assegna aree di RAM per la memorizzazione di dati temporanei, riempie tabelle od altri dati fissi in memoria, ed esegue altre funzioni di governo. Per usare direttive o pseudo-operazioni un programmatore pone il mnemonico della pseudo-operazione nel campo del codice di operazione, ed un indirizzo o dati nel campo dell'indirizzo, se la particolare pseudo-operazione lo richiede.

Le più comuni pseudo-operazioni sono:

DATA
EQUATE o DEFINE
ORIGIN
RESERVE

Assemblatori diversi usano nomi diversi per queste operazioni ma gli scopi sono gli stessi. Le pseudo-operazioni di governo comprendono:

END
LIST
NAME
PAGE
SPACE
TITLE

Queste pseudo-operazioni verranno discusse brevemente, essendo le loro funzioni normalmente ovvie.

LA PSEUDO-OPERAZIONE DATA

La pseudo-operazione DATA permette al programmatore di inviare dati nella memoria di programma. Questi dati possono comprendere:

- Tabelle di consultazione
- Tabelle della conversione in codice
- Messaggi
- Schemi di sincronizzazione
- Soglie
- Nomi
- Coefficienti di equazioni
- Comandi
- Fattori di conversione
- Fattori peso
- Frequenze e tempi caratteristici
- Indirizzi di subroutine
- Identificazione di tasti
- Modelli di prova
- Modelli per la generazione di caratteri
- Modelli di identificazione
- Tabella imposte
- Forme standard
- Pattern o modelli di mascheratura

La pseudo-operazione DATA considera i dati come una parte permanente del programma.

Il formato della pseudo-operazione DATA è normalmente abbastanza semplice. Una istruzione come:

DZCON DATA 12

sostituirà il numero 12 nella successiva locazione di memoria disponibile ed assegnerà alla locazione il nome DZCON. Normalmente ogni pseudo-operazione DATA ha una label a meno che non appartenga ad una serie di pseudo-operazioni DATA. I dati e la label possono assumere qualsiasi forma permessa dall'assemblatore.
Molti assemblatori permettono istruzioni DATA più elaborate che trattano una grande quantità di dati alla volta, per esempio:

```
EMESS   DATA   'ERROR'
SORS    DATA   1,4,9,16,25
```

Una singola istruzione può caricare molte parole nella memoria di programma, con il limite im-

posto dalla sola lunghezza della riga. Si noti che se non si riesce a comprendere tutti i dati in una riga è possibile farla seguire da un'altra con una ulteriore istruzione DATA, per esempio:

```
MESSG   DATA    'ORA È IL'
        DATA    'TEMPO PER TUTTI'
        DATA    'GLI UOMINI BUONI'
        DATA    'DI VENIRE IN'
        DATA    'AIUTO ALLA PROPRIA'
        DATA    'NAZIONE'
```

Gli assemblatori del microprocessore normalmente presentano alcune variazioni rispetto alle pseudo-operazioni DATA standard.
SI DEFINISCE BYTE O BYTE DI FORMATO STANDARD l'insieme di 8 bit; SI DEFINISCE PAROLA O PAROLA DI FORMATO STANDARD l'insieme di 16 bit di numeri o indirizzi. Altre pseudo-operazioni speciali possono trattare dati in carattere codificato.

LA PSEUDO-OPERAZIONE EQUATE (ovvero EQUALS)

DEFINIZIONE NOMI

La pseudo-operazione EQUATE permette al programmatore di uguagliare label e nomi con indirizzi o dati. Questa pseudo-operazione è quasi sempre data dal mnemonico EQU. I nomi possono riferirsi agli indirizzi del dispositivo, dati numerici, indirizzi di partenza, indirizzi fissi, ecc..

La pseudo-operazione EQUATE assegna il valore numerico nel suo campo dell'operando e la label nel suo campo della label. Ecco due esempi:

```
TTY     EQU     5
LAST    EQU     5000
```

La maggior parte degli assemblatori permette di definire una label nei termini di un'altra, per esempio:

```
LAST    EQU     FINAL
ST1     EQU     START + 1
```

La label nel campo dell'operando deve, naturalmente, essere stata definita precedentemente. Spesso il campo dell'operando può contenere espressioni più complesse, come si vedrà più avanti. L'assegnazione di due nomi (due nomi per gli stessi dati o indirizzo) può essere comoda per mettere assieme programmi che usano nomi diversi per la stessa variabile (o diversa ortografia di quello che si riteneva lo stesso nome).

TABELLA DEI SIMBOLI

Si noti che una operazione EQU di un assemblatore non introduce nulla nella memoria. Essa pone semplicemente un nome addizionale in una tabella (detta tabella dei simboli) che l'assemblatore conserva. Questa tabella, diversamente da quella dei mnemonici, deve essere formata da RAM poichè essa è diversa per ogni programma. L'assemblatore di programma richiederà sempre alcune RAM per conservare la tabella dei simboli, più RAM possiede e più simboli può accettare. Questa memoria si aggiunge a quella temporanea richiesta dall'assemblatore.

USO DEI NOMI

Quando si usa un nome? Tutte le volte che si ha un parametro da paragonare al suo ordinario valore numerico ovvero il valore numerico del parametro deve essere cambiato. Tipicamente si assegnano nomi alle costanti di tempo, indirizzi di dispositivi, modelli di mascheratura, fattori di conversione, e simili. Un nome come DELAY, TTY, KBD, NROW, od OPEN non solo rendono più facile il cambiamento del parametro, ma aggiungono anche documentazione al programma. Si assegnano nomi anche alle locazioni di

memoria che hanno scopi speciali; esse possono conservare dati, indicare l'inizio del programma, od essere disponibili per una memorizzazione intermedia.

SCELTA DEI NOMI

Quale nome usare? Le regole migliori sono le stesse delle label soltanto che qui valgono realmente per nomi significativi. Perchè si chiama la telescrivente TTY anzichè X15, il tempo di ritardo di un bit BTIME o BTDLY piuttosto che WW, il numero del tasto «GO» di una tastiera GOKEY piuttosto che HORSE? Questo consiglio sembra banale ma è sorprendente il numero di programmatori che non lo segue.

POSIZIONAMENTO DELLE DEFINIZIONI

Dove è conveniente posizionare le pseudo-operazioni EQUATE? Il miglior posizionamento è all'inizio del programma, sotto appropriati titoli di commento come INDIRIZZI I/O, MEMORIZZAZIONE TEMPORANEA, COSTANTI DI TEMPO, LOCAZIONI DEL PROGRAMMA. Questo rende facile la ricerca della definizione, nel caso si voglia cambiarla. Inoltre un altro utente potrà consultare tutte le definizioni in una zona centralizzata. Chiaramente questa pratica migliora la documentazione e rende il programma più facile da usare.
Le definizioni usate solo in una specifica subroutine potrebbero apparire all'inizio della stessa subroutine.

LA PSEUDO-OPERAZIONE ORIGIN

La pseudo-operazione ORIGIN (quasi sempre abbreviata ORG) permette al programmatore di posizionare, dovunque nella memoria, programmi, subroutine o dati. I programmi ed i dati possono essere posizionati in diverse aree di memoria, in dipendenza della configurazione di quest'ultima. Routine di avviamento, routine di servizio interrupt ed altri programmi richiesti possono essere posizionati in memoria in adatti Indirizzi.

CONTATORE DI LOCAZIONE

L'assemblatore conserva un Contatore di Locazione (paragonabile con il Contatore di Programma del computer) che contiene la locazione di memoria della prossima istruzione o dato che verrà elaborato. Una pseudo-operazione ORG fa in modo che l'assemblatore ponga un nuovo valore nel Contatore di Locazione, come una istruzione di salto fa in modo che la CPU ponga un nuovo valore nel Contatore di Programma. Le uscite dall'assemblatore non devono solo contenere istruzioni o dati ma indicare anche, al caricatore del programma, il posizionamento in memoria delle istruzioni e dei dati.

I programmi per microprocessore spesso contengono statement ORIGIN per i seguenti scopi:

Indirizzo del Reset (Partenza)
Indirizzi del servizio Interrupt
Indirizzi Trap
Memoria RAM
Stack di memoria

Inoltre altri statement ORIGIN possono consentire la formazione di spazio per ulteriori inserzioni, posizionare tabella o dati in memoria, od assegnare le RAM vacanti per il contenimento dei dati. La memoria di programmi e dei dati nei microcomputer possono occupare indirizzi molto sparsi.

Statement ORIGIN tipici sono:

```
ORIG    RESET
ORIG    1000
ORIG    INT 3
```

Alcuni assemblatori assumono un'origine per lo zero, se il programmatore non pone uno state-

ment ORIG all'inizio del programma. Non c'è molta convenienza ma si raccomanda l'uso di uno statement ORG per evitare confusione.

LA PSEUDO-OPERAZIONE RESERVE

La pseudo-operazione RESERVE permette al programmatore di allocare le RAM per vari scopi come tabelle di dati, memorizzazione temporanea, indirizzi indiretti, uno Stack ecc..

ALLOCAZIONE DI RAM

Impiegando la pseudo-operazione RESERVE si assegna un nome all'area di memoria e si dichiara il numero di locazioni di memoria assegnata. Ecco alcuni esempi:

```
NOKEY   RESERVE  1
TEMP    RESERVE  50
VOLTG   RESERVE  80
BUFR    RESERVE  100
```

Si può usare la pseudo-operazione RESERVE per riservare locazioni di memoria nella memoria di programma o dei dati; comunque la natura della pseudo-operazione RESERVE è più significativa quando è applicata alla memoria dati.
In realtà tutte le pseudo-operazioni RESERVE non aumentano il Contatore di Locazione dell'assemblatore oltre la quantità dichiarata nel campo dell'operando. L'assemblatore attualmente non produce alcun codice oggetto.

Si notino le seguenti caratteristiche della pseudo-operazione RESERVE:

1) La label della pseudo-operazione RESERVE ha il valore del primo indirizzo riservato. Per esempio la sequenza:

```
        ORG      3000
BUF 1   RESERVE  100
BUF 2   RESERVE  50
VOLTS   RESERVE  5
```

   assegna alla label BUF 1 il valore 3000, a BUF 2 3100 ed a VOLTS 3150.
2) Occorre specificare il numero di locazioni da riservare. Non esistono eccezioni.
3) Nessun dato viene posizionato nelle locazioni riservate. Qualsiasi dato che, per scambio, può essere posto in queste locazioni ci rimarrà.

INIZIALIZZAZIONE DELLE RAM

Alcuni assemblatori permettono al programmatore di posizionare il valore iniziale delle RAM. Si raccomanda vivamente di non usare questa caratteristica. Si assume che il programma (assieme ai valori iniziali) sarà caricato da un dispositivo esterno (per esempio nastro perforato o floppy disk), ogni volta che si vuole eseguire. La maggior parte di programmi per microprocessore, d'altra parte, risiede in ROM non volatili e diventano operativi con l'inserimento dell'alimentazione. La RAM in tali situazioni non conserva il suo contenuto, nemmeno se è ricaricata. Si aggiungano sempre istruzioni per l'inizializzazione delle RAM nel programma.

PSEUDO-OPERAZIONI DI GOVERNO

Esistono varie pseudo-operazioni di governo che influenzano le funzioni dell'assemblatore e la lista del suo programma piuttosto che il programma di uscita stesso. Le più comuni pseudo-operazioni di governo comprendono:

1) END, che indica la fine del programma sorgente in linguaggio assembly.

2) LIST, che indica all'assemblatore di stampare il programma sorgente. Alcuni assemblatori permettono variazioni come NO LIST o LIST SYMBOL TABLE per evitare, liste lunghe e ripetitive.
3) NAME o TITLE che stampa un nome all'inizio di ogni pagina della lista.
4) PAGE o SPACE che fa saltare alla prossima pagina o riga, rispettivamente, e migliora l'aspetto della lista, rendendo più facile la lettura.
5) PUNCH, che trasferisce il codice oggetto successivo al nastro di carta perforata. Questa pseudo-operazione può in qualche caso far perdere in generalità e perciò non è conveniente.

LABEL CON PSEUDO-OPERAZIONI

Gli utenti spesso trascurano se o quando si può assegnare una label con una pseudo-operazione. Si raccomanda quanto segue:

1) Tutte le pseudo-operazioni EQUATE devono avere la label; diversamente esse perdono significato.
2) Le pseudo-operazioni DATA e RESERVE normalmente hanno la label. La label identifica la prima locazione di memoria utilizzata o assegnata.
3) Le altre pseudo-operazioni potrebbero non avere label. Alcuni assemblatori consentono alle altre pseudo-operazioni di avere label ma il significato di quest'ultima è diverso. Si raccomanda di evitare questa pratica.

INDIRIZZI E CAMPO DELL'OPERANDO

La maggior parte degli assemblatori consente al programmatore una serie di privilegi nella descrizione del contenuto del campo dell'Indirizzo dell'Operando. Ma si ricordi, l'assemblatore ha, per ogni registro, un nome intrinseco e le istruzioni possono avere altri nomi intrinseci.

Alcune opzioni comuni per il campo dell'operando sono:

DATI DECIMALI O INDIRIZZI

1) Numeri decimali:

La maggior parte degli assemblatori assume che tutti i numeri sono decimali se non è specificato diversamente. Così

```
ADD     100
```

significa «aggiunge il contenuto della locazione di memoria 100 decimale al contenuto dell'Accumulatore».

2) Altri sistemi di numeri

ALTRI SISTEMI DI NUMERI

La maggior parte degli assemblatori accettano anche ingressi binari, ottali od esadecimali. Ma occorre identificare questi sistemi di numeri in qualche modo, per esempio facendo precedere o seguire il numero da un carattere identificatore o lettera. Esistono alcuni identificatori comuni:

B per binario.
O, Q o C per ottale (si eviti O perchè fa confusione con zero).
H per esadecimale (o BCD standard).
D per decimale. D può essere omesso, è il caso difettivo.

Gli assemblatori generalmente richiedono numeri esadecimali inizianti con un digit (per esempio, DA_{36} invece di A_{36}) in modo da distinguere numeri e nomi o label. È buona norma introdurre numeri nella base in cui il loro significato è più chiaro — cioè costanti decimali in de-

cimali; indirizzi e numeri BCD in esadecimali; masking pattern o bit di uscita in binario se sono corti, ed in esadecimale se sono lunghi.

3) Nomi simbolici

I nomi possono apparire nel campo dell'operando; essi saranno trattati come i dati che rappresentano. Ma, si ricordi, c'è una differenza tra dati ed indirizzi. La sequenza:

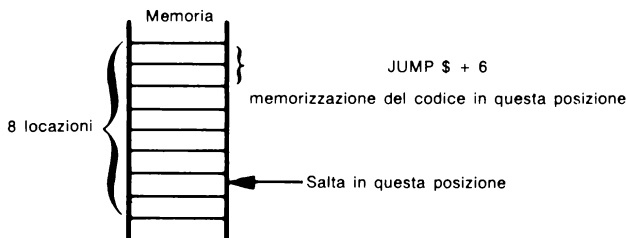
```
FIVE    EQU    5
        ADD    FIVE
```

sommerà il contenuto della locazione di memoria 5 (non necessariamente il numero 5) al contenuto dell'Accumulatore.

4) Il valore corrente della locazione del contatore (normalmente indicato con * o $).

Questo è comodo essenzialmente per una istruzione di Salto, per esempio:

```
JUMP    $ + 6
```

origina un Salto alla locazione di memoria 6 parole oltre la parola che contiene il primo byte dell'istruzione JUMP.

La maggior parte di microprocessori hanno molte istruzioni di due o tre istruzioni. Così si avranno delle difficoltà nella determinazione esatta di quanto possono essere diversi due statement in linguaggio assembly. Perciò basandosi sul valore del Contatore di Locazione spesso si risolve in errori che invece possono essere evitati usando label.

5) Codici dei caratteri

CARATTERI ASCII

La maggior parte degli assemblatori permette al testo di entrare come stringhe ASCII. Queste stringhe possono anche essere circondate con singole o doppie virgolette; le stringhe possono anche essere usate per iniziare a terminare simboli come A o C. Solo pochi assemblatori permettono anche le stringhe EBCDIC. Si raccomanda di usare stringhe di caratteri per tutto il testo. Questo migliora la chiarezza e la leggibilità del programma.

6) Combinazioni delle voci da 1) a 5) operatori, aritmetici, logici o speciali.

ESPRESSIONI ARITMETICHE E LOGICHE

Quasi tutti gli assemblatori permettono semplici combinazioni aritmetiche come START + 1. Alcuni assemblatori permettono la moltiplicazione, divisione, funzioni logiche, spostamenti, ecc.. Queste sono considerate come espressioni. Si noti che l'assemblatore valuta l'espressione nel tempo di assembly. Benchè una espressione nel campo dell'operando possa comprendere una moltiplicazione, può darsi che non si riesca ad usare una moltiplicazione nella logica del programma — a meno che non si scriva una subroutine per lo scopo specifico.

Gli assemblatori si differenziano nella espressione che accettano e come la interpretano. Le espressioni complesse rendono un programma difficoltoso da leggere e da capire.
Sono state fatte alcune raccomandazioni, in questo paragrafo, ma si ritiene opportuno ripeterle ed aggiungerne altre. In generale l'utente dovrebbe esaltare la chiarezza e la semplicità. Non c'è la resa dei conti per un esperto nelle intricazioni degli assemblatori o per la maggior parte delle espressioni complesse del blocco. Si suggerisce il seguente approccio:

1) Si usi il sistema di numeri più chiaro o codice di caratteri per i dati.
2) Le maschere ed i numeri BCD in decimale, i caratteri ASCII in ottale, e le costanti numeriche ordinarie in esadecimale non servono allo scopo e non dovrebbero essere usate.
3) Si ricordi di distinguere i dati e gli indirizzi.
4) Non si usino offset dal Contatore di Locazione.
5) Si mantengano le espressioni semplici ed ovvie. Non fidarsi delle caratteristiche oscure dell'assemblatore.

ASSEMBLY CONDIZIONALE

Alcuni assemblatori permettono di includere parti di programma sorgente, dipendentemente dalle condizioni di esistenza del tempo di assembly. Questo è chiamato assembly condizionale; esso dà all'assemblatore alcune caratteristiche di flessibilità del compilatore. **La maggior parte degli assemblatori per microcomputer hanno possibilità limitate per l'assembly condizionale. Una forma usuale è:**

```
IF COND
•
PROGRAMMA CONDIZIONALE
•
•
•
ENDIF
```

Se l'espressione COND è vera nel tempo di assembly, le istruzioni fra IF ed INDIF (due pseudo-operazioni) sono comprese nel programma.

Impieghi tipici dell'assembly condizionale sono:

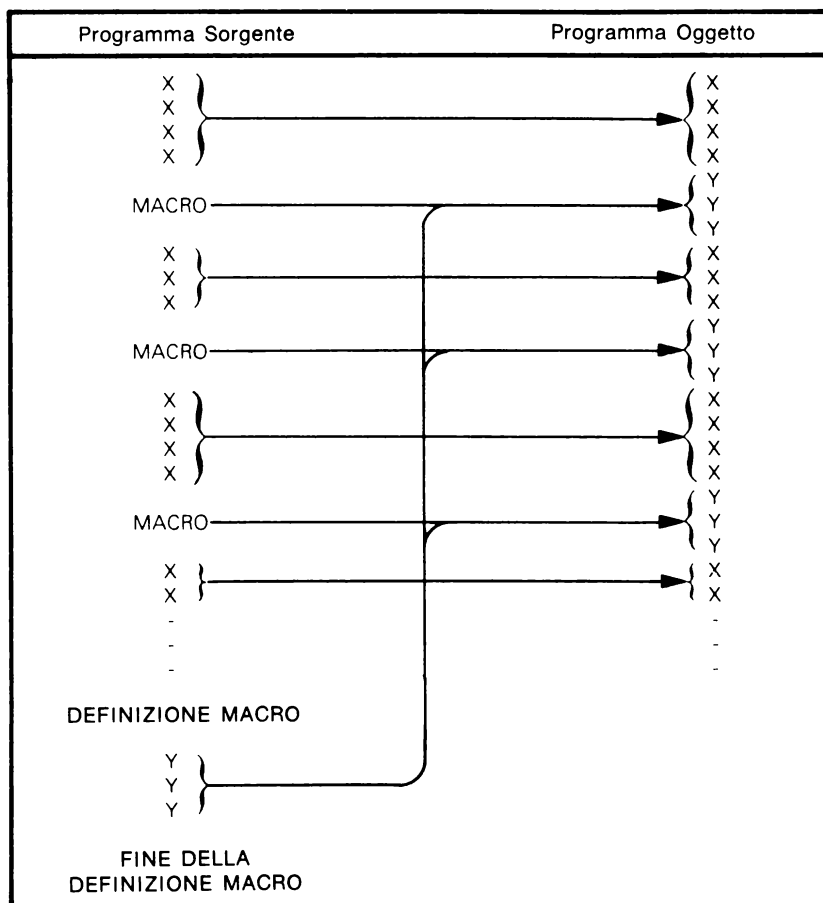
1) Per includere od escludere ulteriori variabili.
2) Per introdurre diagnostici nel testo da eseguire.
3) Per permettere l'uso di dati con diversa lunghezza di bit.

Sfortunatamente, l'assembly condizionale tende ad introdurre disordine nel programma ed a renderlo difficoltoso alla lettura. Si usi quindi l'assembly condizionale solo se è strettamente necessario.

LE MACRO

DEFINIZIONE DI UNA SEQUENZA DI ISTRUZIONI

Si troverà spesso che particolari sequenze di istruzioni possono ricorrere molte volte in un programma sorgente. Sequenze ripetute di istruzioni possono riflettere la richiesta di una logica nel programma ovvero possono essere compensazioni per le deficienze del set di istruzioni del microprocessore. Si possono evitare scritture ripetute della stessa sequenza di istruzioni mediante l'impiego delle macro.

Le macro permettono di assegnare un nome ad una sequenza di istruzioni. Si può poi usare il nome della macro nel programma sorgente, invece di ripetere la sequenza di istruzioni. L'assemblatore sostituirà il nome della macro con l'appropriata sequenza di istruzioni. Questo può essere illustrato come segue:

Le macro non sono la stessa cosa delle subroutine. Una subroutine ricorre una volta in un programma e l'esecuzione del programma si espande nella subroutine. Una macro è espansa in una sequenza di istruzioni ricorrenti ogni volta che la macro è richiamata; così una macro non causa nessuna ramificazione.

Le macro hanno i seguenti vantaggi

VANTAGGI DELLE MACRO

1) Programmi sorgenti più brevi.
2) Migliore documentazione del programma.
3) Uso di sequenze di istruzioni debugged — una volta che una macro è stata collaudata si è sicuri che si usa una sequenza di istruzioni libera da errori ogni volta che si richiama la macro.
4) Più facili cambiamenti. Si cambia la definizione della macro e l'assemblatore esegue il cambiamento ogni volta che la macro è usata.
5) Inclusione di comandi, tasti, od altre istruzioni del computer in un set di istruzioni di base. Si usa quindi la macro come un'estensione del set di istruzione.

Gli svantaggi delle macro sono:

SVANTAGGI DELLE MACRO

1) Ripetizione della stessa sequenza di istruzioni.
2) Una singola macro può creare quantità di istruzioni.
3) Mancanza di standardizzazione.
4) Possibili effetti sui registri e flag che possono non essere chiaramente dichiarati.

VARIABILI LOCALI O GLOBALI

Un problema è che le variabili usate in una macro sono note solo dentro a questa (cioè sono locali piuttosto che globali). Questo può creare spesso una grande confusione senza nessun guadagno che la compensi. È necessario essere consci di questo problema quando si usano le macro.

COMMENTI

Tutti gli assemblatori permettono di posizionare commenti in un programma sorgente. I commenti non hanno effetto sul codice oggetto, ma aiutano a leggere, capire, documentare il programma. Dei buoni commenti sono una parte essenziale della scrittura dei programmi in linguaggio assembly; i programmi senza commenti sono molto difficili da capire.

TECNICHE DI COMMENTO

Si discuterà dei commenti con documentazione in un successivo capitolo, qui vengono riportate alcune indicazioni:

1) Si usino i commenti per dire cosa sta facendo il programma e non cosa fa la singola istruzione.
   I commenti potrebbero dire cose come «LA SUA TEMPERATURA È SOPRA IL LIMITE?», «RIGA CHE ALIMENTA LA TTY», oppure «ESAMINA L'INTERRUTTORE DI CARICO».
   I commenti non dovrebbero dire cose come «AGGIUNGI 1 ALL'ACCUMULATORE», «SALTA ALLO START», oppure «GUARDA IL CARRY». Si dovrebbe descrivere come il programma sta influenzando il sistema; gli effetti interni della CPU hanno raramente qualche interesse.
2) Si usino commenti brevi e sull'aspetto essenziale. I dettagli potrebbero essere disponibili altrove nella documentazione.
3) Si commentino tutti i punti chiave.
4) Non si commentino istruzioni standard o sequenze che cambiano i contatori ed i puntatori; si usi particolare attenzione per le istruzioni che possono avere un significato non ovvio.
5) Non si usino abbreviazioni oscure.
6) I commenti devono essere chiari e leggibili.
7) Si commentino tutte le difinizioni, descrivendo i loro scopi. Si scelgano anche tutte le tabelle e le aree di memoria dati.

8) Si commentino le porzioni del programma come pure le singole istruzioni.
9) Si usi consistenza nei termini. Si può (potrebbe) essere ripetitivi; per evitarlo si consulti un dizionario di sinonimi.
10) Si mettano delle note nei punti in cui si trova confusione, per esempio «RICORDA CHE CARRY È STATO IMPOSTO CON L'ULTIMA ISTRUZIONE». Si può mettere questa nella documentazione finale.

Un programma ben commentato è facile da usare. Si recupererà molte volte il tempo speso nella stesura dei commenti. Si cercherà di mostrare un buon stile di stesura dei commenti negli esempi di programmazione benchè spesso si aggiungeranno quelli per scopi di istruzione.

TIPI DI ASSEMBLATORI

Sebbene tutti gli assemblatori eseguano lo stesso compito, la loro realizzazione è molto variabile. Non si cercherà di descrivere tutti i tipi di assemblatori esistenti; si definiranno soltanto i termini e si indicherà qualche scelta.

CROSS-ASSEMBLATORI

Un cross-assemblatore è un assemblatore che si svolge in un altro computer piuttosto che in quello per il quale assembla i programmi oggetto.
Il computer sul quale l'assemblatore si svolge è tipicamente un grosso computer con costosi supporti di software e periferiche veloci. Esempi tipici sono un IBM 360 o 370, un Univac 1108, ovvero un Burroughs 6700. Il computer per il quale il cross-assemblatore assembla i programmi è tipicamente un micro come l'Intel 8080 od il Motorola 6800.

ASSEMBLATORE RESIDENTE

Un auto-assemblatore, od assemblatore residente, è un assemblatore che si svolge sul computer per il quale assembla i programmi. L'auto-assemblatore richiederà diverse memorie e periferiche ed esso può lavorare abbastanza lentamente.

MACRO-ASSEMBLATORI

Un macro-assemblatore è un assemblatore che permette di definire sequenze di istruzioni come MACRO.

MICRO-ASSEMBLATORI

Un micro-assemblatore è un assemblatore usato per scrivere i microprogrammi che definiscono il set di istruzione di un computer. La microprogrammazione non ha specificatamente a che fare con microcomputer. La microprogrammazione è descritta concettualmente in «AN INTRODUCTION TO MICROCOMPUTERS: Volume I — Basic Concepts», Capitolo 4.

META-ASSEMBLATORI

Un meta-assemblatore è un assemblatore che può maneggiare molti set di istruzione diversi. L'utente deve definire il particolare set di istruzione da usare.

ASSEMBLATORI AD UN PASSO

Un assemblatore ad un passo è un assemblatore che esegue completamente un programma in linguaggio assembly una sola volta. Un tale assemblatore deve avere vie per giungere direttamente alle reference, per esempio istruzioni di salto che usano label che appaiono in seguito nel programma sorgente e che quindi non sono ancora state definite.

ASSEMBLATORI A DUE PASSI

Un assemblatore a due passi è un assemblatore che esegue completamente un programma in linguaggio assembly due volte. La prima volta l'assemblatore semplicemente raccoglie e definisce tutti i simboli; la seconda volta sostituisce le reference con le definizioni reali. Un assemblatore a due passi non ha problemi con reference dirette ma può essere abbastanza lento se una memoria backup (come un floppy disk) non è disponibile. Poi l'assemblatore deve leggere fisicamente il

programma due volte da una unità di ingresso lenta (come una telescrivente o lettore a nastro perforato). La maggior parte degli assemblatori basati sul microprocessore richiedono due passi.

ERRORI

Gli assemblatori normalmente forniscono segnalazione di errore, spesso consistente in una singola lettera codificata. Alcuni errori tipici sono:

1) Nome indefinito (spesso una compilazione incompleta od una definizione omessa).
2) Carattere non consentito (per esempio un 2 in un numero binario).
3) Formato non corretto (delimitatore errato od operandi non corretti).
4) Espressione non valida (per esempio due operatori in una riga).
5) Valore non consentito (normalmente troppo largo).
6) Operando omesso.
7) Doppia definizione (cioè due valori diversi assegnati ad un nome).
8) Label non consentita (per esempio una label in una pesudo-operazione che non può averla).
9) Omissione della label.
10) Codice di operazione non definito.

Per l'interpretazione di errori dell'assemblatore si deve ricordare che l'assemblatore può assumere una pista errata se trova una lettera fuori posto, uno spazio in più, una punteggiatura non corretta. La maggior parte degli assemblatori proseguirà poi nell'interpretazione di qualche stantement successivo producendo messaggi completamente errati. Osservando attentamente il primo errore si può risalire a tutti gli altri derivati da questo. Precauzione, notevole coerenza ed i formati standard elimineranno molti errori fastidiosi.

CARICATORI

Il Caricatore ha la funzione di prelevare il programma (in codice oggetto) dall'assemblatore e posizionato nella memoria. Il caricatore può essere da molto semplice a molto complicato. Se ne descriveranno alcuni tipi diversi.

CARICATORE BOOSTRAP

Un caricatore boostrap è un programma che utilizza una prima parte delle sue istruzioni per caricare le rimanenti o un altro caricatore di programma nella memoria. Il caricatore boostrap può essere in ROM oppure avere l'ingresso nella memoria di un computer tramite un interruttore posto sul fronte del pannello. L'assemblatore può posizionare un caricatore boostrap all'inizio del programma oggetto che produce.

CARICATORE A RILOCAZIONE

Un caricatore a rilocazione può caricare i programmi in qualsiasi posizione della memoria. Esso tipicamente carica ogni programma nello spazio di memoria immediatamente seguente quello usato da programmi precedenti. I programmi comunque devono poter essere rimossi nello stesso modo, cioè, devono essere dotati di rilocazione. Un caricatore assoluto, in contrapposizione, posizionerà i programmi sempre nella stessa area di memoria.

CARICATORI A COLLEGAMENTO

Un caricatore a collegamento carica programmi e subroutine come moduli separati; esso risolve le cross-reference — che è una istruzione in un modulo che si riferisce ad una label in un altro modulo. I programmi oggetto caricati da un caricatore a collegamento devono essere creati da un assemblatore che permetta ed operi con cross-reference.

Capitolo 3

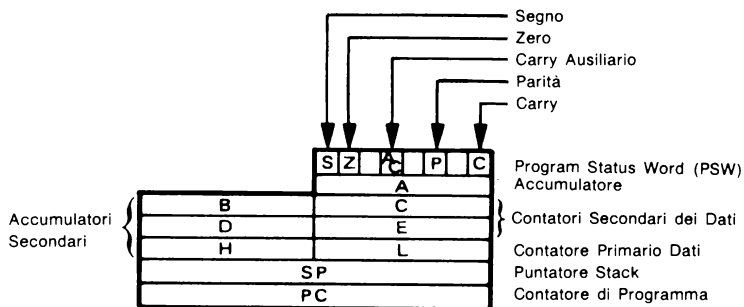
SET D'ISTRUZIONI PER IL LINGUAGGIO ASSEMBLY DELL'8080A ED 8085

A questo punto si può iniziare la costituzione di programmi in linguaggio assembly. Si inizia questo capitolo definendo le singole istruzioni dei set di istruzioni del linguaggio assembly dell'8080A ed 8085, più le regole sintattiche degli assemblatori Intel.

I set di istruzioni per microprocessori 8080A ed 8085 sono identici a parte due istruzioni addizionali (RIM e SIM), disponibili solo sull'8085. Ci sono anche alcune differenze nei cicli di esecuzione di istruzione. La Tabella 3-5 evidenzia queste differenze.

In questo libro non si discuterà nessun aspetto dell'hardware del microcomputer, come pure i segnali, le interfacce ó l'architettura della CPU. Queste informazioni sono riportate in dettaglio in An Introduction To Microcomputers: Volume II – Some Real Products mentre 8080: Programmazione per il Progetto Logico discute il linguaggio assembly come una estensione della logica digitale. In questo libro **si è rivolti alle tecniche di programmazione dal punto di vista del programmatore in linguaggio assembly, dove i pin, ed i segnali sono irrilevanti e quindi non sono importanti le differenze tra un minicomputer ed un microcomputer.**
Interrupt, Acceso Diretto in Memoria, ed architetture dello Stack per l'8080A ed 8085 saranno descritti in un successivo capitolo di questo libro, assieme alla discussione sulla rispettiva programmazione in linguaggio assembly.
La definizione del set di istruzioni data in questo capitolo consiste in una dettagliata definizione di ogni istruzione del linguaggio assembly. Queste definizioni sono identiche a quelle che si trovano sul Capitolo 6 di 8080: Programmazione per il Progetto Logico, eccetto per le due nuove istruzioni dell'8085 (RIM e SIM). La descrizione dettagliata delle singole istruzioni è preceduta da una descrizione generale del set d'istruzioni dell'8080A ed 8085 che divide le istruzioni nei tipi usati comunemente, qualche volta ed usati raramente. Se si ha una esperienza di programmatore in linguaggio assembly questa distinzione non è particolarmente importante ed, in dipendenza dei propri pregiudizi di programmazione, può non essere accurata. Se invece ci si accosta alla programmazione in linguaggio assembly, si raccomanda di usare solo le istruzioni della categoria «comunemente usata». Una volta noti a fondo i concetti della programmazione in linguaggio assembly si possono esaminare le altre istruzioni ed usarle dove sono appropriate.

REGISTRI DELLA CPU E FLAG DI STATO

I registri della CPU ed i flag di Stato sono identici per le CPU dell'8080A ed 8085. I registri ed i flag di posizione possono essere illustrati come segue:

L'Accumulatore è la sorgente primaria e la destinazione per un operando e due istruzioni di operando. Per esempio tutti i trasferimenti di dati tra la CPU ed i dispositivi I/O sono eseguiti attraverso l'Accumulatore. Inoltre molte istruzioni di reference della memoria scambiano dati tra l'Accumulatore e la memoria piuttosto che tra un qualunque altro registro e la memoria. Tutte le istruzioni aritmetiche e Booleane posizionano uno degli operandi nell'Accumulatore ed il risultato si forma nell'Accumulatore stesso. Una istruzione deve perciò **caricare l'Accumulatore prima che l'8080 possa eseguire qualsiasi operazione artimetica o Booleana.**

I registri B, C, D, E, H ed L sono tutti registri secondari. Si può avere accesso con la stessa priorità ai dati memorizzati in questi sei registri; tali dati possono essere messi in ogni altro registro o possono essere usati come secondo operando nelle istruzioni a due operandi.
Ci sono, comunque, alcune differenze importanti nelle funzioni dei Registri, B, C, D, E, H ed L.

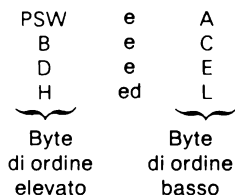
I registri H ed L sono i Puntatori dei Dati per l'8080A ed 8085. Occorre dire che normalmente si useranno questi registri per conservare gli indirizzi di memoria a 16 bit di dati a cui si vuole accedere. I dati possono essere trasferiti tra qualsiasi registro e la locazione di memoria indirizzata da H ed L. Usando qualsiasi altro metodo di indirizzamento di memoria, solo l'Accumulatore può agire come sorgente o destinazione di dati dentro la CPU. Perciò i programmatori dell'8080A ed 8085 possono indirizzare ai dati della memoria attraverso i registri H ed L ogni volta che è possibile.

Dentro la logica del programma si riservi sempre i registri H ed L per contenere l'indirizzo dei dati della memoria.

I registri B, C, D ed E provvedono alla memorizzazione secondaria dei dati. Frequentemente il secondo operando di una istruzione a due operandi è memorizzato in uno di questi quattro registri. (Il primo operando è memorizzato nell'Accumulatore che è anche la destinazione del risultato).

C'è un numero limitato di istruzioni che tratti i registri B e C o D ed E come Puntatori di Dati a 16 bit. Ma queste istruzioni muovono i dati soltanto tra la memoria e l'Accumulatore.

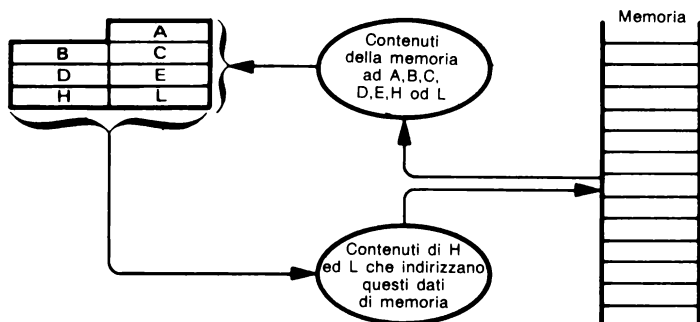
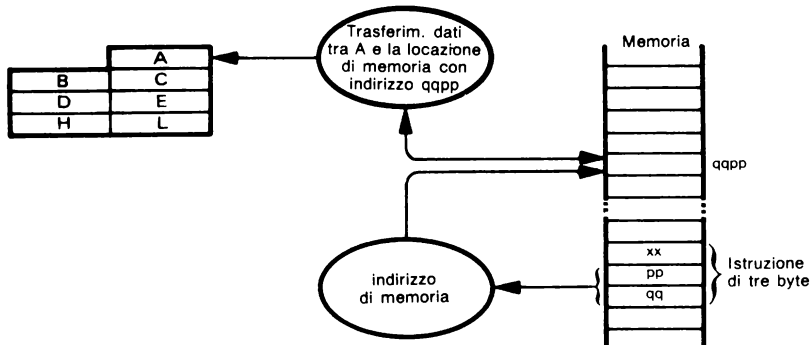
In una logica di programma si dovrebbe normalmente usare i registri B, C, D ed E per la memorizzazione temporanea di dati ed indirizzi.

Esiste un certo numero di istruzioni che usa dati di 16 bit alla volta. Queste istruzioni fanno riferimento alle coppie di registri della CPU come segue:

La combinazione dell'Accumulatore e del Program Status Word, considerata come unità a 16 bit, è usata solo per operazioni dello Stack. Le operazioni aritmetiche accedono a B e C o D ed E o H ed L come unità dati a 16 bit.

Il flag del Carry estrae il bit più significativo di qualsiasi operazione aritmetica. Il flag del Carry è anche compreso in istruzioni di spostamento; esso è ripristinato mediante istruzioni Booleane.

Il flag indicatore di Parità è posto al livello logico 1 quando un'operazione aritmetica o Booleana produce un risultato che contiene un numero pari di bit 1.
È posto al livello logico zero quando una operazione produce un risultato con un numero dispari di bit 1.
Il flag Zero è posto ad 1 quando qualsiasi operazione aritmetica o Booleana genera come risultato 0. Lo stato Zero è posto a 0 quando tale operazione genera un risultato diverso da zero.
Il flag indicatore Segno acquista il valore del bit più significativo del risultato seguente l'esecuzione di qualsiasi istruzione aritmetica o Booleana.
Il flag indicatore Carry Ausiliario conserva i bit 3 e 4 di qualsiasi Carry risultante dell'esecuzione di una istruzione aritmetica. Lo scopo di questo flag indicatore è di semplificare le operazioni del Decimale-Codifcato-Binario (BCD); questo è l'uso standard di un flag indicatore Carry Ausiliario come descritto in An Introduction To Microcomputers: Volume I, Capitolo 3.
Il Puntatore dello Stack a 16 bit permette di realizzare uno stack di ogni luogo della memoria indirizzabile — la dimensione dello stack è limitata soltanto dalla quantità presente di memoria indirizzabile. In realtà si può veramente usare più di 256 byte di memoria per uno stack. Si potrebbe usare lo stack per l'indirizzamento di subroutine e la elaborazione di interrupt. Non si usi lo stack per fornire i parametri alle subroutine. Questo non è molto efficiente nelle limitazioni del set di istruzioni dell'8080A. Lo stack 8080A/8085 viene iniziato dal suo più alto indirizzo. Un Push decrementa il contenuto del Puntatore dello Stack, un Pop lo incrementa.

INDIRIZZAMENTO DELLA MEMORIA PER L'8080A/8085

Quando si procede all'indirizzamento dei dati in memoria occorre usare l'indirizzamento implicato o quello diretto. Nell'indirizzamento implicato della memoria, i registri H ed L contengono gli indirizzi dei registri di memoria a cui si deve accedere (ovvero li identificano). I dati possono essere mossi tra le locazioni di memoria identificate ed uno qualsiasi dei registri A, B, C, D, E, H o L della CPU. Questo **può essere illustrato come segue:**

Un numero limitato di istruzioni usa i registri B e C, o D ed E come Puntatori dei Dati. Queste istruzioni muovono i dati tra l'Accumulatore e la locazione di memoria indirizzata dai Registri B e C od i Registri D ed E.
Esiste anche un limitato numero di istruzioni che usa l'indirizzamento diretto della memoria per trasferire i dati tra l'Accumulatore e la memoria. Ci sono istruzioni di tre byte che possono essere illustrate come segue:

Poichè lo stack è una parte della memoria di Lettura e Scrittura occorre considerare le istruzioni di Push e Pop come istruzioni di reference della memoria. Queste istruzioni muovono due byte di dati tra una coppia di registri e l'indirizzo del Puntatore dello Stack, cioè la sommità dello stack corrente. L'indirizzo dello stack 8080A/8085 è decrementato con ogni Push ed incrementato con ogni Pop.

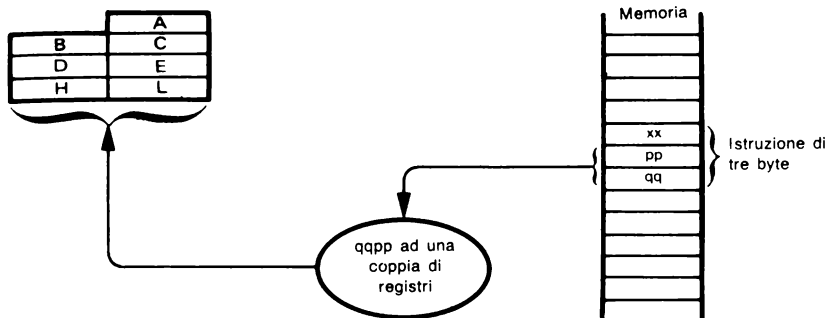
Alcuni testi identificano le istruzioni immediate con istruzioni di Reference della memoria. Una istruzione immediata è una istruzione di due o tre byte nella quale il secondo e terzo byte contengono dati fissi che sono caricati nell'Accumulatore od alcuni altri registri della CPU. Questo può essere illustrato come segue:

Tutte le istruzioni di Salto dell'8080A/8085 usano l'indirizzamento assoluto, diretto; questo è come dire che tutte le istruzioni di salto sono lunghe tre byte, in cui il secondo e terzo byte contengono l'indirizzo della locazione di memoria dalla quale il microprocessore prenderà la prossima istruzione da eseguire.

Tabella 3-1
Istruzioni Dell'8080A/8085 Frequentemente Usate

Codice Istruzione	Significato
ADC, ACI	SOMMA CON CARRY
ADD, ADI	SOMMA
ANA, ANI	AND LOGICO
CALL	CHIAMA SUBROUTINE
CMP, CPI	PARAGONA
DCR	DECREMENTA
IN	INGRESSO
INR	INCREMENTA
INX	INCREMENTA 16 BIT
JC	SALTA SE CARRY
JMP	SALTA
JNC	SALTA SE NON CARRY
JNZ	SALTA SE NON ZERO
JZ	SALTA SE ZERO
LDA	CARICA ACCUMULATORE
LXI	CARICA 16 BIT
MOV	MUOVI
MVI	MUOVI DIRETTO
OUT	USCITA
RAL	RUOTA CON CARRY SINISTRA
RAR	RUOTA CON CARRY DESTRA
RET	RITORNA DALLA SUBROUTINE
STA	MEMORIZZA ACCUMULATORE
SUB, SUI	SOTTRAI

Tabella 3-2
Istruzioni Dell'8080A/8085 Occasionalmente Usate

Codice Istruzione	Significato
CMA	COMPLEMENTA ACCUMULATORE
DAA	AGGIUSTA DECIMALI
DAD	SOMMA 16 BIT
DCX	DECREMENTA 16 BIT
DI	DISABILITA INTERRUPT
EI	ABILITA INTERRUPT
HLT	FERMATI
JM	SALTA SE MENO
JP	SALTA SE POSITIVO
LDAX	CARICA ACCUMULATORE INDIRETTO
LHLD	CARICA H ED L DIRETTO
NOP	NO OPERAZIONE
ORA, ORI	OR LOGICO
POP	RIMUOVI DALLO STACK
PUSH	ENTRA NELLO STACK
RIM (8085 ONLY)	RIPRISTINA INTERRUPT MASK
RLC	RUOTA SINISTRA
RRC	RUOTA DESTRA
SHLD	MEMORIZZA H ED L DIRETTO
SIM (8085 ONLY)	POSIZIONA INTERRUPT MASK
STAX	MEMORIZZA ACCUMULATORE INDIRETTO
XCHG	SCAMBIA D ED E, H ED L
XRA, XRI	OR ESCLUSIVO LOGICO

Tabella 3-3
Istruzioni Dell'8080A/8085 Raramente Usate

Codice Istruzione	Significato
CC	CHIAMA SE CARRY
CM	CHIAMA SE MENO
CMC	COMPLEMENTA CARRY
CNC	CHIAMA SE NO CARRY
CNZ	CHIAMA SE NO ZERO
CP	CHIAMA SE POSITIVO
CPE	CHIAMA SE PARITÀ PARI
CPO	CHIAMA SE PARITÀ DISPARI
CZ	CHIAMA SE ZERO
JPE	SALTA SE PARITÀ PARI
JPO	CHIAMA SE PARITÀ DISPARI
PCHL	H ED L AL CONTATORE PROGRAMMA
RC	RITORNA SE CARRY
RM	RITORNA SE MENO
RNC	RITORNA SE NO CARRY
RNZ	RITORNA SE NO ZERO
RP	RITORNA SE NO POSITIVO
RPE	RITORNA SE PARITÀ PARI
RPO	RITORNA SE PARITÀ DISPARI
RST	RIPARTI
RZ	RITORNA SE ZERO
SBB, SBI	RITORNA CON PRESTITO
SPHL	H ED L AL PUNTATORE STACK
STC	PONI CARRY
XTHL	SCAMBIA SOMMITÀ STACK, H ED L

Le istruzioni erroneamente spaventano gli utenti nuovi alla programmazione. Considerate come un evento isolato, le operazioni associate con l'esecuzione di una singola i-struzione sono abbastanza facili da eseguire — e questo è lo scopo del presente capitolo.

Perchè le istruzioni di un microcomputer vengono considerate come un «set» di istruzioni? Perché le istruzioni selezionate dai progettisti di qualunque microcomputer sono selezionate con grande cura; è molto più facile eseguire operazioni complesse come una sequenza di semplici eventi — ognuno dei quali è rappresentato da una istruzione appartenente ad un ben definito «set» d'istruzioni.

In accordo con An Introduction To Microcomputers: Volume II, la Tabella 3-4 che riassume il set d'istruzioni 8080A/8085, nella quale istruzioni simili sono raggruppate assieme. Le singole istruzioni sono descritte in seguito in ordine alfabetico di istruzione mnemonica.

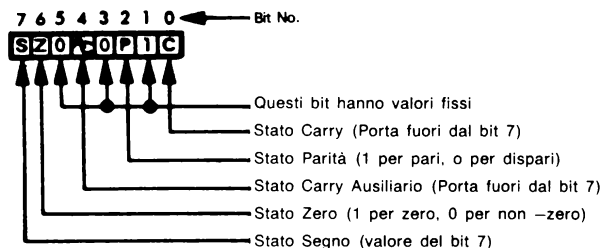
Inoltre per semplificare la dichiarazione fatta da ogni istruzione viene identificato lo scopo del-l'istruzione all'interno della normale logica di programmazione.

ABBREVIAZIONI

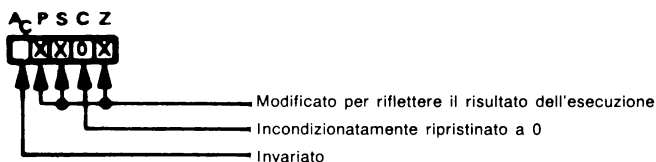
Queste sono le abbreviazioni usate in questo capitolo:

A	L'Accumulatore
B	Il registro B } Questi sono a volte considerati come una coppia di registri
C	Il registro C
D	Il registro D } Questi sono a volte considerati come una coppia di registri
E	Il registro E
H	Il registro H } Questa coppia di registri fornisce l'indirizzamento implicato della memoria
L	Il registro L
CS	Stato Carry
Ac	Stato Carry Ausiliario
ZS	Stato Zero
SS	Stato Segno
PS	Stato Parità
I	Il registro di istruzione
I2	Secondo byte codice oggetto
I3	Terzo byte codice oggetto
PC	Il Contatore di Programma
SP	Il Puntatore dello Stack
PSW	Il Program Status Word che ha bit assegnati ai flag indicatori come mostrato alla pagina successiva
H	Comparendo alla fine di un gruppo di digit (per esempio 213 AH) sta a designare i digit esadecimali
dati	Dati ad 8 bit immediati
dev	Un dispositivo I/O
dati 16	Dati a 16 bit immediati
reg	Registri A, B, C, D, E, H od L
M	Memoria, indirizzo implicato da HL
label	Un indirizzo a 16 bit, specificando una label di istruzione
rp	Una coppia di registri: B per BC, D per DE, H per HL, SP per Puntatore dello Stack, PSW per flag indicatori ed Accumulatore
port	Una porta I/O, identificata da un numero tra 0 ed FF_{16}
addr	Un indirizzo a 16 bit, specificando un byte, della memoria dati
[ ]	Contenuti della locazione identificata dentro la parentesi
[[ ]]	Parola della memoria indirizzata dalla locazione identificata dentro le parentesi
←	Muovi i dati in direzione della freccia
←→	Scambia i contenuti delle locazioni ad entrambi i lati della freccia
+	Somma
−	Sottrai
•, ∧	AND
V	OR
⊕, $\veebar$	XOR

STATO

I cinque flag di stato sono memorizzati in un registro Program Status Zord (PSW) come segue:

PSW ed A sono talvolta trattati come una coppia di registri.

L'effetto dell'esecuzione di un'istruzione sullo stato è illustrato come segue:

CAMBIAMENTI DELLO STATO CON L'ESECUZIONE DI ISTRUZIONE

Nelle illustrazioni di esecuzione di istruzione, una X identifica uno stato imposto o ripristinato. Uno 0 identifica uno stato che è sempre basso. Un bianco significa che lo stato non cambia.

MNEMONICI DI ISTRUZIONE

La parte fissa di una istruzione in linguaggio assembly è mostrata in ALTO.
La parte variabile (dati immediati, numero dispositivo I/O, nome del registro, label o indirizzo) sono mostrati in basso.

CODICI OGGETTO DI ISTRUZIONE

I codici oggetto delle istruzioni sono rappresentati con due digit esadecimali per istruzioni senza variazioni.
I codici oggetto delle istruzioni sono rappresentati con otto digit binari per istruzioni con variazioni; sono così identificabili le variazioni della rappresentazione in digit binari.

TEMPO DI ESECUZIONE DI ISTRUZIONE E CODICI

La Tabella 3-5 elenca le istruzioni in ordine alfabetico, mostrando i codici oggetto ed i tempi di esecuzione espressi in cicli macchina.
Dove sono mostrati due cicli di istruzione, il primo è per «condizione non soddisfatta» mentre il secondo è per «condizione soddisfatta».

Tabella 3-4
Sommario Del Set Di Istruzioni Del Microcomputer 8080A/8085

Tipo	Mnemonico	Operando/i	Byte	Stato					Operazione Eseguita
				C	AC	Z	S	P	
I/O	IN	DEV	2						[A] ← [DEV] Ingresso da A al dispositivo DEV (DEV da 0 a 255)
	OUT	DEV	2						[DEV] ←[A] Uscita da A al dispositivo DEV (DEV da 0 a 255)
Primaria Reference di Memoria	LDAX	RP	1						[A] ← [[RP]] Carica A usando l'indirizzo implicato da BC (RP = B) o DE (RP = D)
	STAX	RP	1						[[RP]] ← [A] Memorizza A usando l'indirizzamento implicato come per LDAX
	MOV	R,M	1						[R] ← [[HL]] Carica tutti i registri usando l'indirizzo implicato da HL
	MOV	M,R	1						[[HL]] ← [R] Memorizza tutti i registri usando l'indirizzo implicato da HL
	LDA	ADDR	3						[A] ← [ADDR], cioè [A] ← [[I3,I2]] Carica A, usa l'indirizzamento diretto
	STA	ADDR	3						[ADDR] ← [A], cioè [[I3,I2]] ← [A] Carica A, usa l'indirizzamento diretto
	LHLD	ADDR	3						[L] ← [ADDR], [H] ← [ADDR + 1] cioè, [L] ← [[I3,I2]], [H] ← [[I3,I2] + 1] Carica i registri H ed L, usa l'indirizzamento diretto
	SHLD	ADDR	3						[ADDR] ← [L], [ADDR + 1] ← [H] cioè, [[I3,I2]] ← [L], [[I3,I2] + 1] ← [H] Memorizza i registri H ed L, usa l'indirizzamento diretto

Tabella 3-4

Sommario Del Set Di Istruzioni Del Microcomputer 8080A/8085 (continuazione)

Tipo	Mnemonico	Operando/i	Byte	Stato					Operazione Eseguita
				C	AC	Z	S	P	
Secondaria Reference di Memoria Opera Memoria	ADD	M	1	X	X	X	X	X	[A] ← [A] + [[HL]] Somma ad A
	ADC	M	1	X	X	X	X	X	[A] ← [A] + [[HL]] + [CS] Somma ad A con Carry
	SUB	M	1	X	X	X	X	X	[A] ← [A] - [[HL]] Sottrai da A
	SBB	M	1	X	X	X	X	X	[A] ← [A] - [[HL]] - [CS] Sottrai da A con prestito
	ANA	M	1	0	X**	X	X	X	[A] ←[A] Λ[[HL]] AND con A
	XRA	M	1	0	0	X	X	X	[A] ← [A] ⊻ [[HL]] OR – Esclusivo con A
	ORA	M	1	0	0	X	X	X	[A] ← [A] V [[HL]] OR con A
	CMP	M	1	X	X	X	X	X	[A] - [[HL]] Paragona con A
	INR	M	1		X	X	X	X	[[HL]] ← [[HL]] + 1 Incrementa Memoria
	DCR	M	1		X	X	X	X	[[HL]] ← [[HL]] - 1 Decrementa Memoria
Immediato	LXI	RP,DATA16	3						[RP] ←DATA16 Carica dati a 16-bit immediati in BC (RP = B), DE (RP = D), HL (RP = H) o SP (RP = SP)
	MVI	M,DATA	2						[[HL]] ←DATA Carica dati ad 8-bit immediato nella locazione di memoria a-vente indirizzo implicato da HL
	MVI	R,DATA	2						[R] ←DATA Carica dati ad 8-bit immediato in qualsiasi registro

Tabella 3-4
Sommario Del Set Di Istruzioni Del Microcomputer 8080A/8085 (continuazione)

Tipo	Mnemonico	Operando/i	Byte	Stato					Operazione Eseguita
				C	AC	Z	S	P	
Salto	JMP	ADDR	3						[PC] ← ADDR Salta all'istruzione con la label ADDR
	PCHL		1						[PC] ← [HL] Salta all'istruzione con indirizzo contenuto in HL
Chiamata e Ritorno da Subroutine (Immediato e Stack)	CALL	ADDR	3						[[SP]] ← [PC], [PC] ← ADDR, [SP] ← [SP] - 2 Salta alla subroutine iniziante ad ADDR
	CC	ADDR	3						[[SP]] ← [PC], [PC] ← ADDR, [SP] ← [SP] - 2 Salta alla subroutine se C = 1
	CNC	ADDR	3						[[SP]] ← [PC], [PC] ← ADDR, [SP] ← [SP] - 2 Salta alla subroutine se C = 0
	CZ	ADDR	3						[[SP]] ← [PC], [PC] ← ADDR, [SP] ← [SP] - 2 Salta alla subroutine se Z = 1
	CNZ	ADDR	3						[[SP]] ← [PC], [PC] ← ADDR, [SP] ← [SP] - 2 Salta alla subroutine se Z = 0
	CP	ADDR	3						[[SP]] ← [PC], [PC] ← ADDR, [SP] ← [SP] - 2 Salta alla subroutine se S = 0
	CM	ADDR	3						[[SP]] ← [PC], [PC] ← ADDR, [SP] ← [SP] - 2 Salta alla subroutine se S = 1
	CPE	ADDR	3						[[SP]] ← [PC], [PC] ← ADDR, [SP] ← [SP] - 2 Salta alla subroutine se parità pari
	CPO	ADDR	3						[[SP]] ← [PC], [PC] ← ADDR, [SP] ← [SP] - 2 Salta alla subroutine se parità dispari
	RET		1						[PC] ← [[SP]], [SP] ← [SP] + 2 Ritorna dalla subroutine

Tabella 3-4
Sommario Del Set Di Istruzioni Del Microcomputer 8080A/8085 (continuazione)

Tipo	Mnemonico	Operando/i	Byte	Stato					Operazione Eseguita
				C	AC	Z	S	P	
Chiamata e Ritorno da Subroutine (Immediato e Stack)	RC		1						[PC] ← [[SP]], [SP] ← [SP] + 2 Ritorna dalla subroutine se C = 1
	RNC		1						[PC] ← [[SP]], [SP] ← [SP] + 2 Ritorna dalla subroutine se C = 0
	RZ		1						[PC] ← [[SP]], [SP] ← [SP] + 2 Ritorna dalla subroutine se Z = 1
	RNZ		1						[PC] ← [[SP]], [SP] ← [SP] + 2 Ritorna dalla subroutine se Z = 0
	RM		1						[PC] ← [[SP]], [SP] ← [SP] + 2 Ritorna dalla subroutine se S = 1
	RP		1						[PC] ← [[SP]], [SP] ← [SP] + 2 Ritorna dalla subroutine se S = 0
	RPE		1						[PC] ← [[SP]], [SP] ← [SP] + 2 Ritorna dalla subroutine se parità pari
	RPO		1						[PC] ← [[SP]], [SP] ← [SP] + 2 Ritorna dalla subroutine se parità dispari
Opera Immediato	ADI	DATA	2	X	X	X	X	X	[A] ← [A] + DATA Somma immediato ad A
	ACI	DATA	2	X	X	X	X	X	[A] ← [A] + DATA + [CS] Somma con carry immediato ad A
	SUI	DATA	2	X	X	X	X	X	[A] ← [A] - DATA Sottrai immediato da A
	SBI	DATA	2	X	X	X	X	X	[A] ← [A] - DATA - [CS] Sottrai immediato con prestito da A
	ANI	DATA	2	0	X**	X	X	X	[A] ← [A] Λ DATA AND immediato con A

Tabella 3-4

Sommario Del Set Di Istruzioni Del Microcomputer 8080A/8085 (continuazione)

Tipo	Mnemonico	Operando/i	Byte	Stato					Operazione Eseguita
				C	AC	Z	S	P	
Opera Immediato	XRI	DATA	2	0	0	X	X	X	[A] ← [A] ⊻ DATA OR – esclusivo immediato con A
	ORI	DATA	2	0	0	X	X	X	[A] ← [A] V DATA OR immediato con A
	CPI	DATA	2	X	X	X	X	X	[A] - DATA Confronta immediato con A
Salto Condizionato	JC	ADDR	3						[PC] ← ADDR Salta se C = 1
	JNC	ADDR	3						[PC] ← ADDR Salta se C = 0
	JZ	ADDR	3						[PC] ← ADDR Salta se Z = 1
	JNZ	ADDR	3						[PC] ← ADDR Salta se Z = 0
	JP	ADDR	3						[PC] ← ADDR Salta se S = 0
	JM	ADDR	3						[PC] ← ADDR Salta se S = 1
	JPE	ADDR	3						[PC] ← ADDR Salta con parità pari
	JPO	ADDR	3						[PC] ← ADDR Salta con parità dispari

Tabella 3-4

Sommario Del Set Di Istruzioni Del Microcomputer 8080A/8085 (continuazione)

Tipo	Mnemonico	Operando/i	Byte	Stato					Operazione Eseguita
				C	AC	Z	S	P	
Movimento Reg — Reg	MOV	D,S	1						[R] ← [R] Muovi qualsiasi registro (S) in qualunque altro (D)
	XCHG		1						[D] ←→ [H], [E] ←→ [L] Scambia DE con HL
	SPHL		1						[HL] → [SP] Manda HL in SP
Operazioni Registro-Registro	ADD	R	1	X	X	X	X	X	[A] ← [A] + [R] Somma qualunque registro ad A
	ADC	R	1	X	X	X	X	X	[A] ← [A] + [R] + [CS] Somma con Carry qualsiasi registro ad A
	SUB	R	1	X	X	X	X	X	[A] ← [A] - [R] Sottrai qualsiasi registro da A
	SBB	R	1	X	X	X	X	X	[A] ← [A] - [R] - [CS] Sottrai qualsiasi registro con prestito da A
	ANA	R	1	0	X**	X	X	X	[A] ← [A] $\wedge$ [R] AND di qualunque registro con A
	XRA	R	1	0	0	X	X	X	[A] ← [A] $\veebar$ [R] OR — esclusivo di qualunque registro con A
	ORA	R	1	0	0	X	X	X	[A] ← [A] V [R] OR qualunque registro con A
	CMP	R	1	X	X	X	X	X	[A] - [R] Confronta qualunque registro con A
Opera Registro	INR	R	1		X	X	X	X	[R] ← [R] + 1 Incrementa qualsiasi registro
	DCR	R	1		X	X	X	X	[R] ← [R] - 1 Decrementa qualsiasi registro
	CMA		1						[A] ← [$\bar{A}$] Complementa A

Tabella 3-4

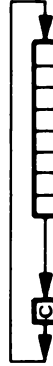
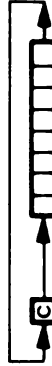
Sommario Del Set Di Istruzioni Del Microcomputer 8080A/8085 (continuazione)

Tipo	Mnemonico	Operando/i	Byte	Stato					Operazione Eseguita
				C	AC	Z	S	P	
Opera Registro	DAA		1	X	X	X	X	X	Aggiusta i decimali di A
	RLC		1	X					Ruota A a sinistra con ramificazione del carry
	RRC		1	X					Ruota A a destra con ramificazione del carry
	RAL		1	X					Ruota A a sinistra con carry
	RAR		1	X					Ruota A a destra con carry
	DAD	RP	1	X					[HL] ← [HL] + [RP] Somma RP ad HL. RP = BC, DE, HL o SP
	INX	RP	1						[RP] ← [RP] + 1 Incrementa RP. RP = BC, DE, HL o SP
	DCX	RP	1						[RP] ← [RP] - 1 Decrementa RP. RP = BC, DE, HL o SP
Stack	PUSH	RP	1						[[SP]] ← [RP], [SP] ← [SP] - 2 Carica il contenuto di RP nello Stack } RP=BC, DE, HL o SP
	POP	RP	1						[RP] ← [[SP]], [SP] ← [SP] + 2 Preleva lo stack in RP } RP=BC, DE, HL o SP
	XTHL		1						[HL] ←→ [[SP]] Scambia HL con sommità dello Stack

Tabella 3-4
Sommario Del Set Di Istruzioni Del Microcomputer 8080A/8085 (continuazione)

Tipo	Mnemonico	Operando/i	Byte	Stato					Operazione Eseguita
				C	A_C	Z	S	P	
Interrupt	EI		1						Abilita interrupt dopo l'esecuzione della prossima istruzione
	RIM		1						Leggi Interrupt Mask*
	DI		1						Disabilita Interrupt
	SIM		1						Poni Interrupt Mask*
	RST		1						Riparti .
Stato	STC		1	1					[CS] ← 1 Poni Carry
	CMC		1	X					[CS] ← [CS] Complementa Carry
	NOP		1						
	HLT		1						

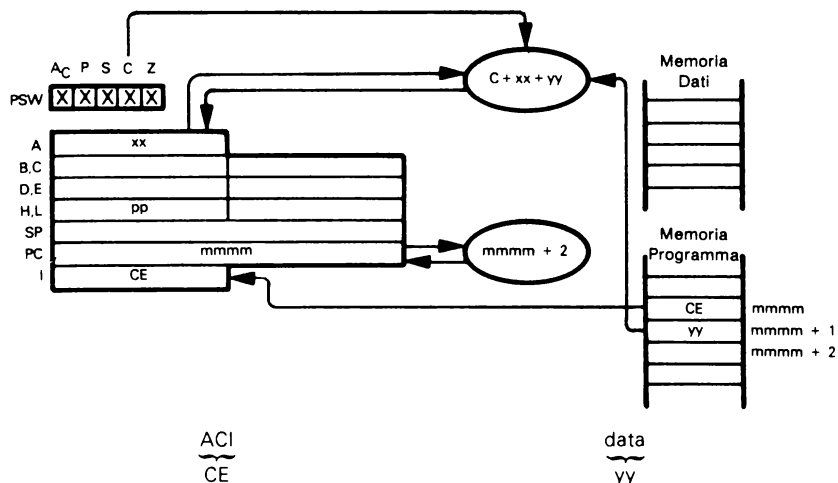
Stati:
C = Carry
A_C = Carry oltre il 3 bit
Z = Zero
S = Segno
P = Parità
X = Stato set o reset
0 = Stato reset

Bianco = Stato invariato
*Istruzioni 8085
*8085 pone A_C ad 1 per tutte le istruzioni AND

Tabella 3-5
Sommario Dei Codici Oggetto Di Istruzione E Cicli Di Esecuzione

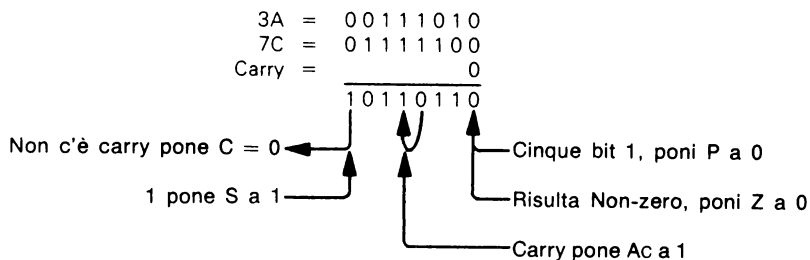
Istruzione	Cod. Oggetto	Byte	Cicli	Istruzione	Cod. Oggetto	Byte	Cicli
ACI DATA	CE yy	2	7	LXI RP,DATA16	00xx0001	3	10
ADC REG	10001xxx	1	4		yyyy		
ADC M	8E	1	7	MOV REG,REG	01dddsss	1	5
ADD REG	10000xxx	1	4	MOV M,REG	01110sss	1	7
ADD M	86	1	7	MOV REG,M	01ddd110	1	7
ADI DATA	C6 yy	2	7	MVI REG,DATA	00ddd110	2	7
ANA REG	10100xxx	1	4		yy		
ANA M	A6	1	7	MVI M,DATA	36 yy	2	10
ANI DATA	E6 yy	2	7	NOP	00	1	4
CALL LABEL	CD ppqq	3	17	ORA REG	10110xxx	1	4
CC LABEL	DC ppqq	3	11/17	ORA M	B6	1	7
CM LABEL	FC ppqq	3	11/17	ORI DATA	F6 yy	2	7
CMA	2F	1	4	OUT PORT	D3 yy	2	10
CMC	3F	1	4	PCHL	E9	1	5
CMP REG	10111xxx	1	4	POP RP	11xx0001	1	10
CMP M	BE	1	7	PUSH RP	11xx0101	1	11
CNC LABEL	D4 ppqq	3	11/17	RAL	17	1	4
CNZ LABEL	C4 ppqq	3	11/17	RAR	1F	1	4
CP LABEL	F4 ppqq	3	11/17	RC	D8	1	5/11
CPE LABEL	EC ppqq	3	11/17	RET	C9	1	10
CPI DATA	FE yy	2	7	RIM	20	1	4*
CPO LABEL	E4 ppqq	3	11/17	RLC	07	1	4
CZ LABEL	CC ppqq	3	11/17	RM	F8	1	5/11
DAA	27	1	4	RNC	D0	1	5/11
DAD RP	00xx1001	1	10	RNZ	C0	1	5/11
DCR REG	00xxx101	1	5	RP	F0	1	5/11
DCR M	35	1	10	RPE	E8	1	5/11
DCX RP	00xx1011	1	5	RPO	E0	1	5/11
DI	F3	1	4	RRC	0F	1	4
EI	FB	1	4	RST N	11xxx111	1	11
HLT	76	1	7	RZ	C8	1	5/11
IN PORT	DB yy	2	10	SBB REG	10011xxx	1	4
INR REG	00xxx100	1	5	SBB M	9E	1	7
INR M	34	1	10	SBI DATA	DE yy	2	7
INX RP	00xx0011	1	5	SHLD ADDR	22 ppqq	3	16
JC LABEL	DA ppqq	3	10	SIM	30	1	4*
JM LABEL	FA ppqq	3	10	SPHL	F9	1	5
JMP LABEL	C3 ppqq	3	10	STA ADDR	32 ppqq	3	13
JNC LABEL	D2 ppqq	3	10	STAX RP	000x0010	1	7
JNZ LABEL	C2 ppqq	3	10	STC	37	1	4
JP LABEL	F2 ppqq	3	10	SUB REG	10010xxx	1	4
JPE LABEL	EA ppqq	3	10	SUB M	96	1	7
JPO LABEL	E2 ppqq	3	10	SUI DATA	D6 yy	2	7
JZ LABEL	CA ppqq	3	10	XCHG	EB	1	4
LDA ADDR	3A ppqq	3	13	XRA REG	10101xxx	1	4
LDAX RP	000x1010	1	7	XRA M	AE	1	7
LHLD ADDR	2A ppqq	3	16	XRI DATA	EE yy	2	7
				XTHL	E3	1	18

ppqq — rappresenta quattro indirizzi di memoria in digit esadecimali
yy — rappresenta due digit di dati esadecimali
yyyy — rappresenta quattro digit di dati esadecimali
x — rappresenta un digit binario opzionale
ddd — rappresenta i digit binari opzionali identificanti un registro di destinazione
sss — rappresenta i digit binari opzionali identificanti un registro sorgente
*Istruzioni 8085

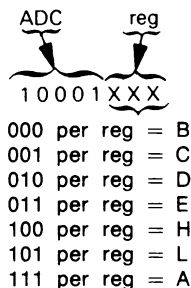
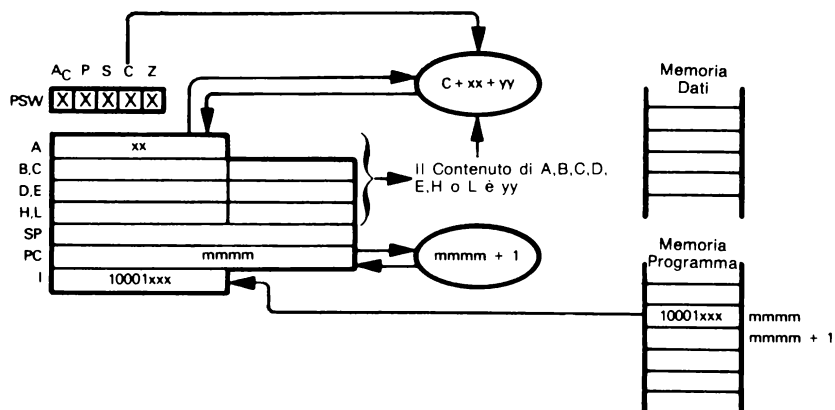
ACI – SOMMA CON CARRY IMMEDIATO ALL'ACCUMULATORE

Somma il contenuto del successivo byte della memoria di programma e lo stato Carry all'Accumulatore.

Si supponga xx = $3A_{16}$, yy = $7C_{16}$, C = 0. Dopo che l'istruzione:

ACI 7CH

è stata eseguita, l'Accumulatore conterrà B6:

Questa è una subroutine di istruzione alla manipolazione dei dati.

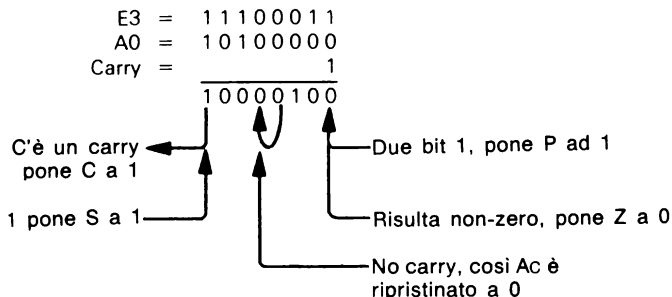
ADC — SOMMA DI UN REGISTRO O MEMORIA CON CARRY ALL'ACCUMULATORE

Questa istruzione assume due forme. La prima considera i contenuti di un registro sommato all'Accumulatore:

Somma il contenuto del registro A, B, C, D, E, H o L e lo stato Carry all'Accumulatore.

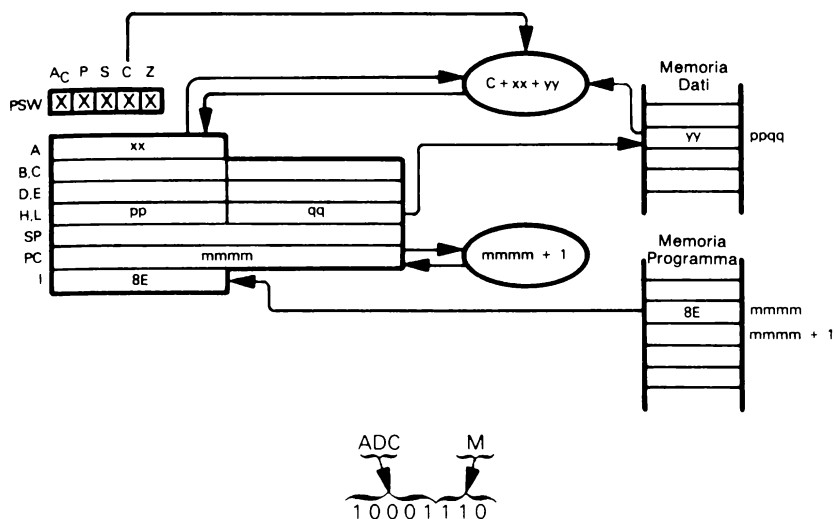
Si supponga xx = $E3_{16}$, il registro E contenga $A0_{16}$, C = 1.

Dopo che l'istruzione:

ADC E

è stata eseguita l'Accumulatore conterrà 84_{16}:

I contenuti del byte di memoria possono anche essere sommati, con Carry, all'Accumulatore:

ADC M

10001110

Se xx = $E3_{16}$, yy = $A0_{16}$ e C = 1, allora l'esecuzione dell'istruzione:

ADC M

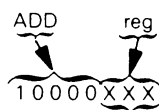
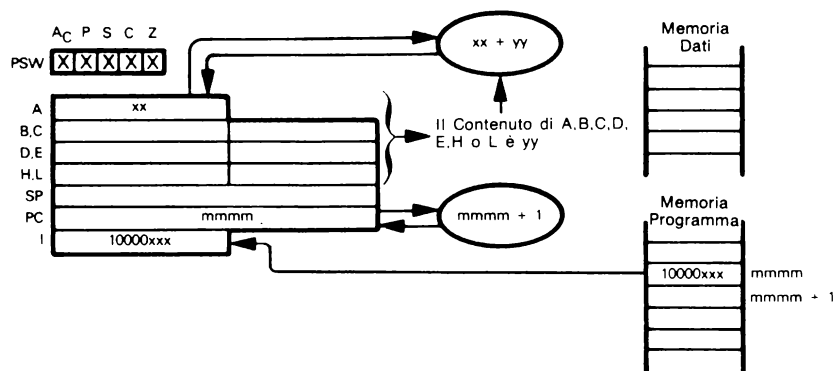
genera lo stesso risultato dell'esecuzione dell'istruzione ADC E appena descritta.

L'istruzione ADC è principalmente usata nell'addizione multibyte, per il secondo e successivi byte.

ADD — SOMMA DI REGISTRO O MEMORIA ALL'ACCUMULATORE

Questa istruzione assume due forme. La prima considera i contenuti di un registro sommati all'Accumulatore:

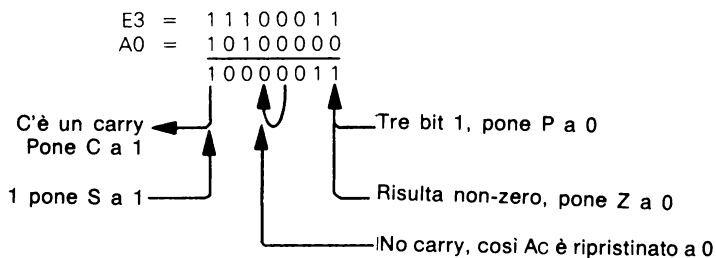
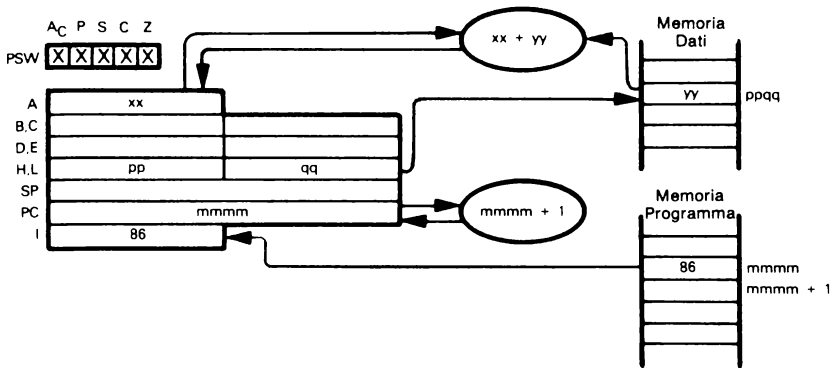
A_C P S C Z
PSW X X X X X
A xx
B,C
D,E
H,L
SP
PC mmmm
I 10000xxx
xx + yy
Il Contenuto di A,B,C,D, E,H o L è yy
mmmm + 1
Memoria Dati
Memoria Programma
10000xxx mmmm
mmmm + 1

ADD reg
1 0 0 0 0 X X X

000 per reg = B
001 per reg = C
010 per reg = D
011 per reg = E
100 per reg = H
101 per reg = L
111 per reg = A

Somma i contenuti del registro, A, B, C, D, E, H o L all'Accumulatore.

Si supponga xx = $E3_{16}$, il registro E contenga $A0_{16}$, C = 1. Dopo che l'istruzione:

ADD E

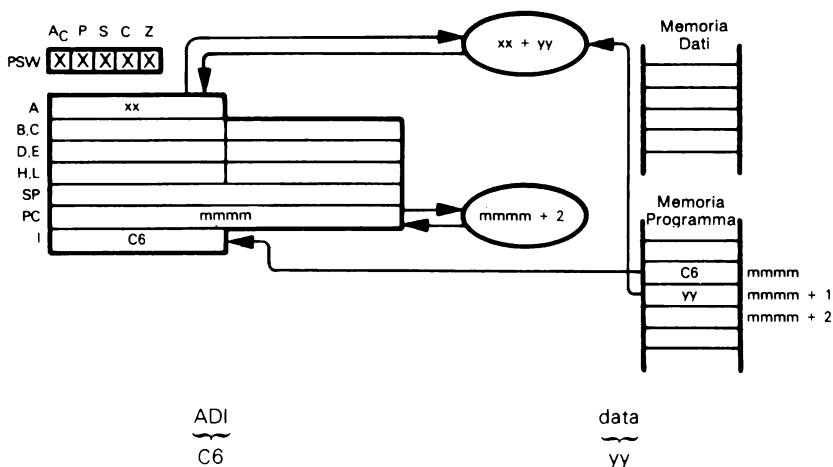
è stata eseguita, l'Accumulatore conterrà 83_{16}:

I contenuti del byte di memoria possono anche essere sommati all'Accumulatore:

Se xx = E3₁₆, yy = A0₁₆ e C = 1 allora l'esecuzione dell'istruzione:

ADD M

genera lo stesso risultato l'esecuzione dell'istruzione ADD E appena descritta.

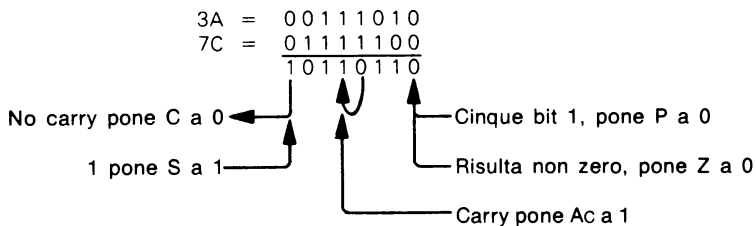
ADD è l'istruzione dell'addizione binaria usata nelle operazioni normali, a singolo byte; è anche l'istruzione usata per sommare i byte di basso ordine di due numeri multi byte.

ADI — SOMMA IMMEDIATA ALL'ACCUMULATORE

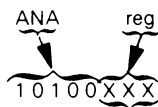
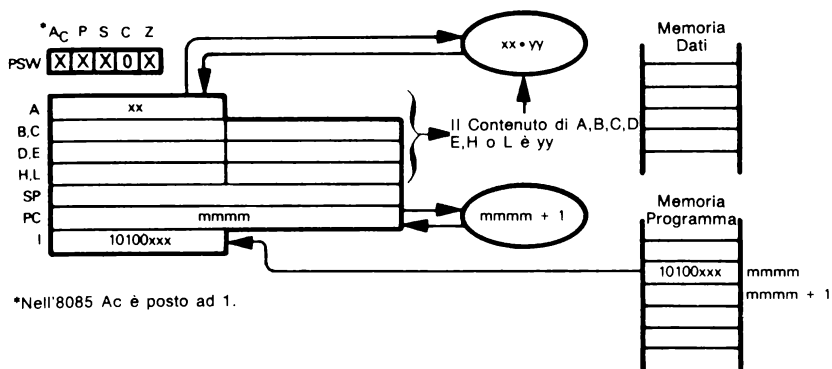
ADI   data
C6   yy

Somma il contenuto del successivo byte della memoria di programma all'Accumulatore.

Si supponga xx = $3A_{16}$, yy = $7C_{16}$, C = 0. Dopo che l'istruzione:

ADI   7CH

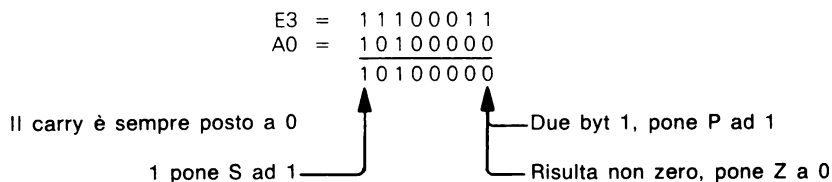
è stata eseguita l'accumulatore conterrà B6:

Questa è una istruzione di manipolazione dati di routine.

ANA – AND DI REGISTRO O MEMORIA CON L'ACCUMULATORE

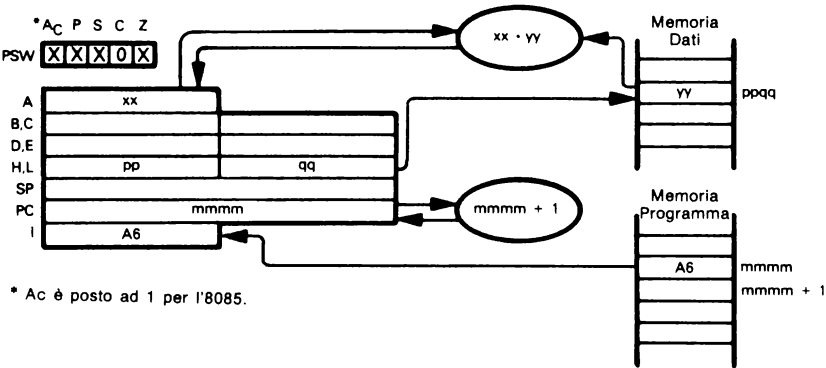
Questa istruzione assume due forme. La prima esegue l'AND fra contenuto del registro e l'Accumulatore:

Opera l'AND tra l'accumulatore ed il registro A, B, C, D, E, H o L ed il risultato resta nell'Accumulatore.

Si supponga xx = $E3_{16}$, il registro contenga $A0_{16}$. Dopo che l'istruzione:

ANA E

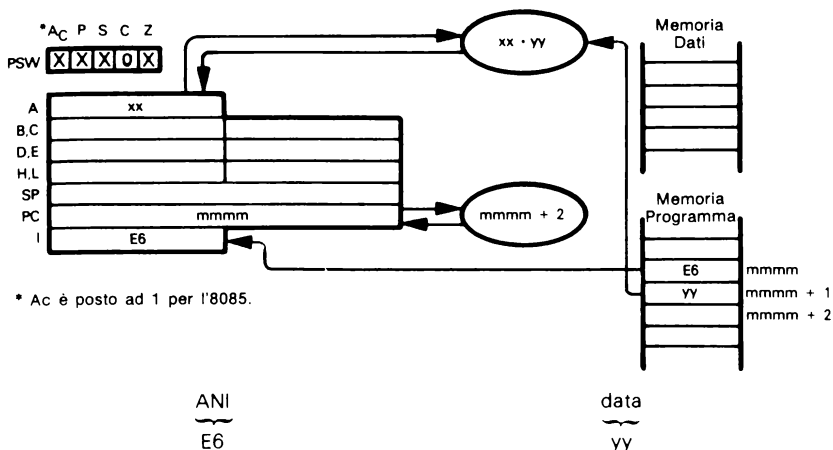
è stata eseguita, l'Accumulatore conterrà $A0_{16}$:

Si può operare l'AND tra i contenuti di un byte di memoria e l'Accumulatore:

ANA M

10100110

Se xx = $E3_{16}$, yy = $A0_{16}$ e C = 1, allora l'esecuzione dell'istruzione:

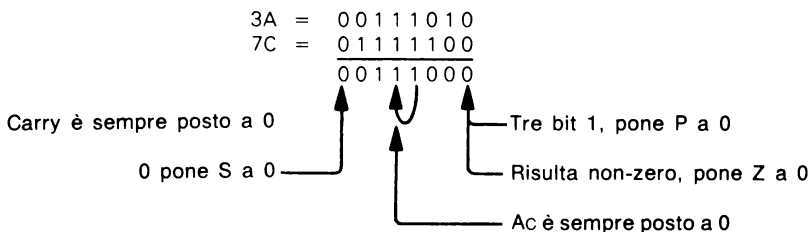
ANA M

genera lo stesso risultato dell'esecuzione dell'istruzione ANA E, appena descritta. ANA è una istruzione logica usata frequentemente.

ANI — AND IMMEDIATO CON L'ACCUMULATORE

Opera l'AND fra il contenuto del successivo byte della memoria di programma e l'Accumulatore.

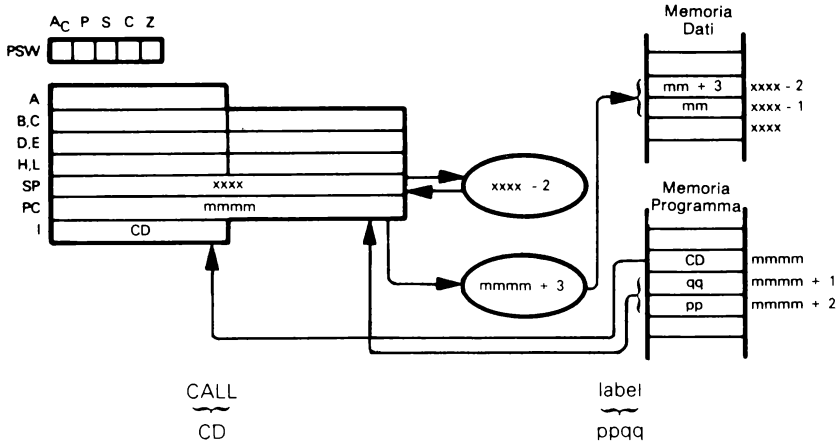
Si supponga xx = $3A_{16}$, yy = $7C_{16}$. Dopo che l'istruzione:

ANI 7CH

è stata eseguita, l'Accumulatore conterrà 38_{16}.

Questa è una istruzione logica di routine; è usata spesso per porre «off» dei bit. Per esempio l'istruzione:

ANI 7FH

porrà incondizionatamente a 0 il bit di ordine elevato dell'Accumulatore.

CALL – CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO

Memorizza l'indirizzo dell'istruzione seguente CALL alla sommità dello stack; la sommità dello stack è un byte della memoria dati, indirizzato dal Puntatore dello Stack. Poi sottrae 2 dal Puntatore dello Stack in modo da indirizzare la nuova sommità dello Stack. Muove l'indirizzo a 16 bit contenuto nel secondo e terzo byte dell'istruzione CALL del programma oggetto e lo riporta nel Contatore di Programma.

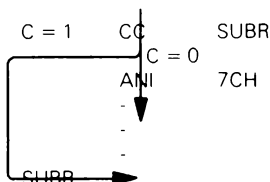
Si consideri la sequenza di istruzioni:

```
        CALL    SUBR
        ANI     7CH
        —
        —
        —
SUBR
```

Dopo che l'istruzione CALL è stata eseguita l'indirizzo dell'istruzione ANI è contenuto alla sommità dello stack. Il Puntatore dello Stack è decrementato di 2. Poi verrà eseguita l'istruzione con la label SUBR.

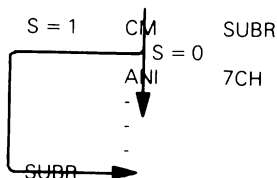
CC — CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLTANTO SE LO STATO CHARRY È UGUALE AD 1

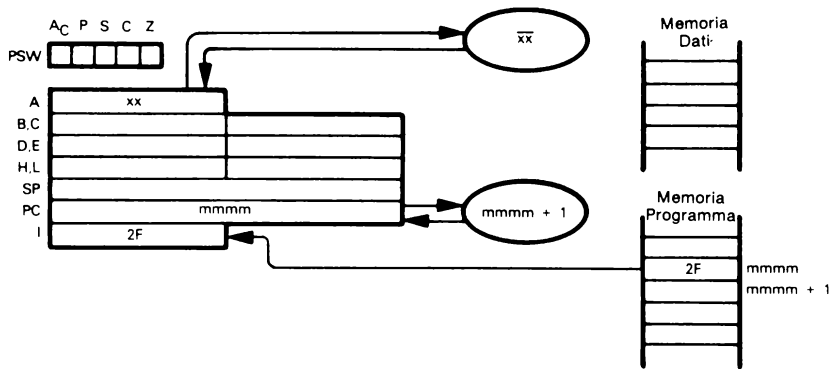
Questa istruzione è identica alla istruzione CALL tranne il fatto che la subroutine identificata sarà chiamata solo se lo stato CARRY è uguale a 1, diversamente sarà eseguita l'istruzione sequenzialmente seguente l'istruzione CC.

Si consideri la seguente sequenza di istruzioni:

Dopo che l'istruzione CC è stata eseguita, se lo stato Carry non è uguale ad 1, verrà eseguita l'istruzione ANI. Se lo stato Carry è uguale ad 1 l'indirizzo della istruzione ANI è conservato alla sommità dello Stack. Il Puntatore dello Stack è decrementato di 2. L'istruzione con la label SUBR sarà la prossima ad essere eseguita.

CM — CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLTANTO SE LO STATO SIGN È UGUALE AD 1

Questa istruzione è identica alla istruzione CALL eccetto che la subroutine identificata sarà chiamata solo se lo stato SIGN è uguale ad 1; diversamente sarà eseguita l'istruzione sequenzialmente seguente l'istruzione CM.

Si consideri la sequenza di istruzioni:

Dopo che è stata eseguita l'istruzione CM, se lo stato Sign non è uguale ad 1, verrà eseguita l'istruzione ANI. Se lo stato Sign è uguale ad 1, l'indirizzo dell'istruzione ANI è conservato alla sommità dello Stack. Il Puntatore dello Stack è decrementato di 2. La prossima istruzione ad essere eseguita sarà quella con la label SUBR.

CMA — COMPLEMENTA L'ACCUMULATORE

$$\underbrace{\text{CMA}}_{\text{2F}}$$

Complementa il contenuto dell'Accumulatore. Non viene influenzato il contenuto di nessun altro registro o stato.

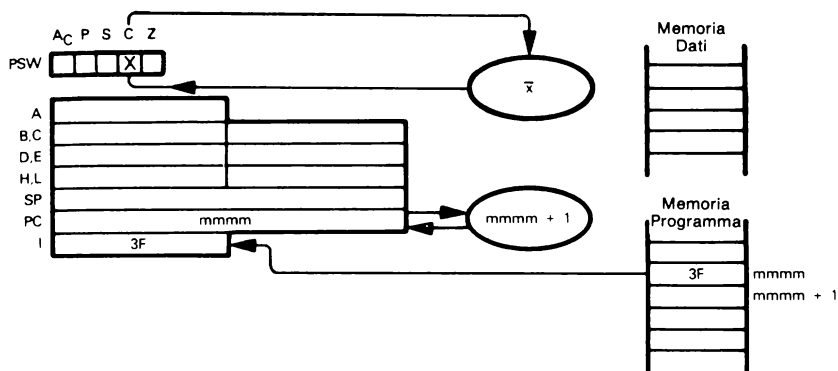
Si suppone che l'Accumulatore contenga $3A_{16}$. Dopo che l'istruzione:

CMA

è stata eseguita, l'Accumulatore conterrà $C5_{16}$.

$$\begin{aligned} 3A_{16} &= 00111010 \\ \text{Complemento} &= 11000101 \end{aligned}$$

Questa è una tipica istruzione logica di routine. **Non si usa per la sottrazione binaria.** Allo scopo esistono istruzioni speciali per la sottrazione (SUB ed SBB).

CMC — COMPLEMENTA LO STATO CARRY

$$\underbrace{\text{CMC}}_{\text{3F}}$$

Complementa lo stato Carry. Non viene influenzato nessun altro stato o contenuto di registro.

Si supponga che lo stato Carry contenga 1. Dopo che l'istruzione:

CMC

è stata eseguita, lo stato Carry conterrà 0.

Questa istruzione viene usata per forzare lo stato Carry a 0, per mezzo della sequenza di istruzioni:

```
STC    ;PONE LO STATO CARRY AD 1
CMC    ;COMPLEMENTA LO STATO CARRY
```

Si noti che si può porre lo stato Carry a 0 per mezzo della istruzione:

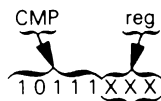
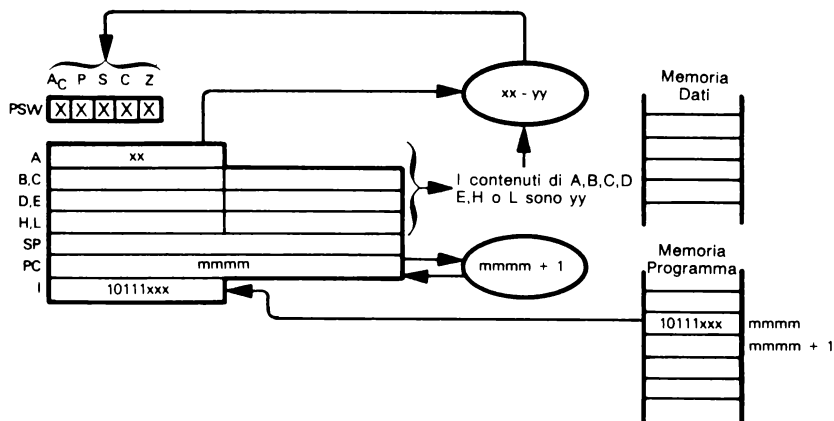
ANA A

che automaticamente azzera lo stato Carry ma non modifica nessun altro contenuto di registro poichè viene eseguito l'AND dell'Accumulatore con sè stesso. L'istruzione:

ORA A

serve allo stesso scopo.

CMP – CONFRONTA REGISTRO O MEMORIA CON L'ACCUMULATORE

Questa istruzione assume due forme. La prima considera i contenuti di un registro confrontati con l'Accumulatore:

Ac P S C Z
PSW X X X X X
A xx
B,C
D,E
H,L
SP
PC mmmm
I 10111xxx
xx - yy
I contenuti di A,B,C,D E,H o L sono yy
mmmm + 1
Memoria Dati
Memoria Programma
10111xxx mmmm
mmmm + 1

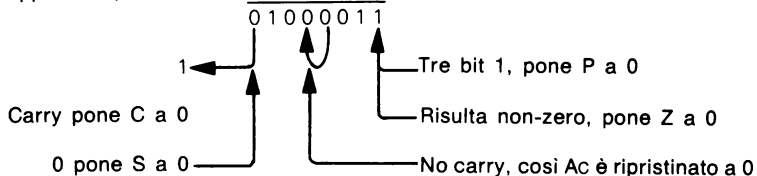
CMP reg
1 0 1 1 1 X X X

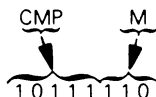
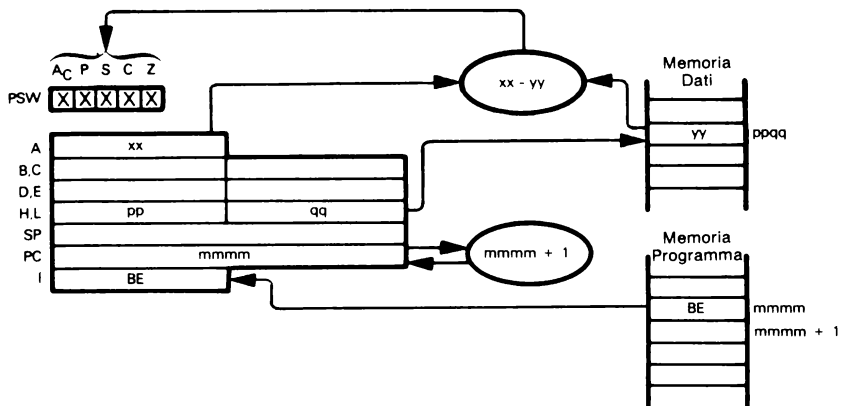
000 per reg = B
001 per reg = C
010 per reg = D
011 per reg = E
100 per reg = H
101 per reg = L
111 per reg = A

Sottrae i contenuti di registro A, B, C, D, E, H o L dal contenuto dell'Accumulatore, considerando entrambi i numeri come semplici dati binari. Viene poi scaricato il risultato, cioè caricato il solo Accumulatore ma modificati i flag di stato per riflettere il risultato della sottrazione.

Si supponga xx = $E3_{16}$, il registro E contenga $A0_{16}$. Dopo che l'istruzione:

CMP E

è stata eseguita l'Accumulatore conterrà $E3_{16}$ ma gli stati saranno modificati come segue:

Si noti che il Carry risulta complementato.

Anche i contenuti di un byte di memoria possono essere paragonati con l'Accumulatore:

Se xx = $E3_{16}$ ed yy = $A0_{16}$, allora la esecuzione dell'istruzione:

CMP M

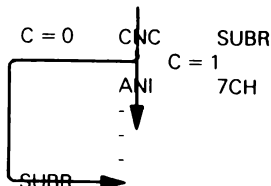
genera lo stesso risultato dell'esecuzione dell'istruzione CMP E, appena descritta.

Le istruzioni di confronto frequentemente precedono le istruzioni di Call condizionale, Ritorno e Salto. L'istruzione Paragona Immediato (CPI) è più pratica dell'istruzione CMP.

CNC — CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLTANTO SE LO STATO CARRY È UGUALE A 0

CNC label
D4 ppqq

Questa istruzione è identica all'istruzione salvo che qui l'istruzione identificata verrà chiamata solo se lo stato Carry è uguale a 0; diversamente verrà eseguita l'istruzione sequenzialmente seguente l'istruzione CNC.

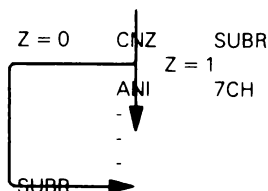
Si consideri la sequenza di istruzioni:

Dopo che è stata eseguita l'istruzione CNC, se lo stato Carry non è uguale a 0 sarà eseguita l'istruzione ANI. Se lo stato Carry è uguale a zero, l'indirizzo dell'istruzione ANI è conservato alla sommità dello Stack. Il Puntatore dello Stack è decrementato di 2. La prossima istruzione ad essere eseguita sarà quella con la label SUBR.

CNZ — CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLTANTO SE LO STATO ZERO È UGUALE A 0

CNZ — label

C4 — ppqq

Questa istruzione è identica all'istruzione CALL eccetto che la subroutine identificata sarà chiamata solo se lo stato Zero è uguale a 0; diversamente sarà eseguita l'istruzione sequenzialmente seguente la CNZ.

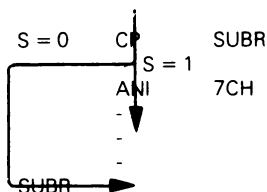
Si consideri la sequenza di istruzioni:

Dopo che è stata eseguita l'istruzione CNZ, se lo stato Zero non è uguale a 0 sarà eseguita l'istruzione ANI. Se lo stato Zero è uguale a 0, l'indirizzo dell'istruzione ANI è conservato alla sommità dello Stack. Il Puntatore dello Stack è decrementato di 2. La prossima istruzione che sarà eseguita è quella avente label SUBR.

CP — CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLTANTO SE LO STATO SIGN È UGUALE A 0

CP — label

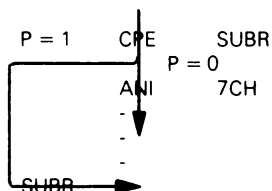
F4 — ppqq

Questa istruzione è identica all'istruzione CALL eccetto che la subroutine identificata sarà chiamata solo se lo stato Sign è uguale a 0; diversamente sarà eseguita l'istruzione sequenzialmente seguente la CP.

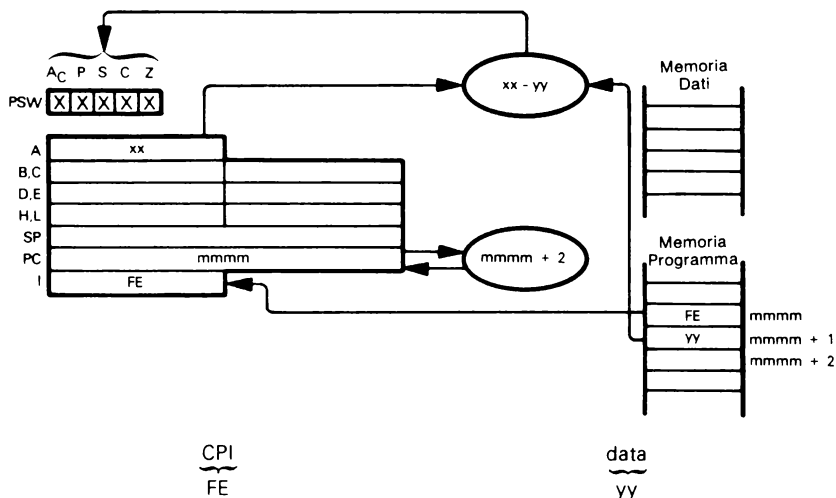
Si consideri la sequenza di istruzioni:

Dopo che è stata eseguita l'istruzione CP, sarà eseguita l'istruzione ANI se lo stato Sign non è uguale a 0. Se lo stato Sign è uguale a 0, l'indirizzo dell'istruzione ANI è conservato alla sommità dello Stack. Il Puntatore dello Stack è decrementato di 2. La prossima istruzione ad essere eseguita sarà quella con la label SUBR.

CPE – CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLTANTO SE LO STATO PARITÀ È UGUALE AD 1

Questa istruzione è identica all'istruzione CALL eccetto che la subroutine identificata può essere chiamata solo se lo stato Parità è uguale ad 1; diversamente sarà eseguita l'istruzione sequenzialmente seguente la CPE.

Si consideri la sequenza di istruzioni:

Dopo che l'istruzione CPE è stata eseguita, se lo stato Parità non è uguale ad 1 sarà eseguita l'istruzione ANI. Se lo stato Parità è uguale ad 1, l'indirizzo dell'istruzione ANI è conservato alla sommità dello stack. Il Puntatore dello Stack è decrementato di 2. La prossima istruzione ad essere eseguita sarà quella con la label SUBR.

CPI — CONFRONTA I CONTENUTI DELL'ACCUMULATORE CON DATI IMMEDIATI

CPI — FE   data — YY

Sottrae i contenuti del secondo byte in codice oggetto dai contenuti dell'Accumulatore trattando entrambi i numeri come singoli dati binari. Scarica il risultato, cioè lascia invariato l'Accumulatore, ma modifica i flag di Stato per riflettere il risultato della sottrazione.

Si supponga xx = $E3_{16}$ ed il secondo byte dell'istruzione CPI in codice oggetto contiene $A0_{16}$. Dopo che l'istruzione:

CPI   A0H

è stata eseguita, l'Accumulatore conterrà $E3_{16}$ ma gli stati saranno modificati come segue:

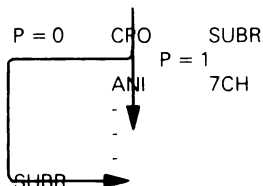
Si noti che il Carry risultante è complementato.

Questa è l'istruzione in gran parte usata per imporre gli stati precedenti l'esecuzione di un'istruzione di chiamata convenzionale, Ritorno o Salto.

CPO — CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLTANTO SE LO STATO PARITÀ È UGUALE A 0

CPO	label
E4	ppqq

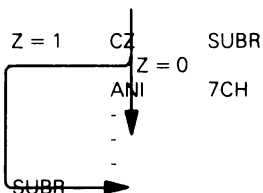
Questa istruzione è identica all'istruzione CALL eccetto che la subroutine identificata sarà richiamata solo se lo stato Parità è uguale a 0; diversamente sarà eseguita l'istruzione sequenzialmente seguente la CPO.

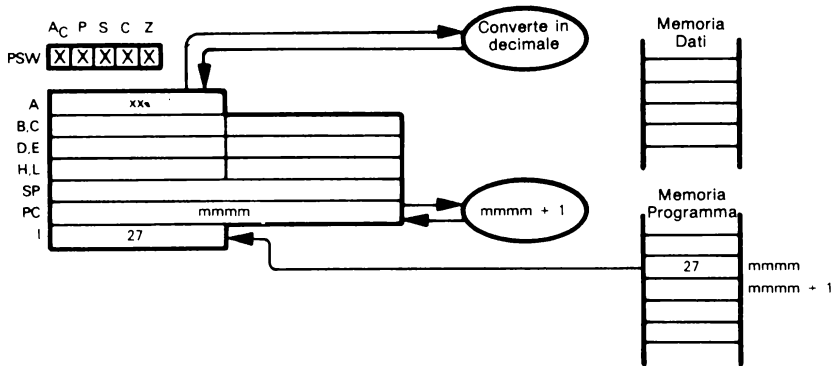
Si consideri la sequenza di istruzioni:

Dopo che è stata eseguita l'istruzione CPO, se lo stato Parità non è uguale a 0 sarà eseguita l'istruzione ANI. Se lo stato Parità è uguale a 0, l'indirizzo dell'istruzione ANI è conservato alla sommità dello stack. Il Puntatore dello Stack è decrementato di 2. La successiva istruzione ad essere eseguita sarà quella con la label SUBR.

CZ — CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLO SE LO STATO ZERO È UGUALE AD 1

CZ	label
CC	ppqq

Questa istruzione è identica all'istruzione CALL eccetto che la subroutine identificata sarà richiamata solo se lo stato Zero è uguale ad 1; diversamente sarà eseguita l'istruzione sequenzialmente seguente la CZ.

Si consideri la sequenza di istruzioni:

Dopo che l'istruzione CZ è stata eseguita, se lo stato Zero non è uguale a 1 sarà eseguita l'istruzione ANI. Se lo stato Zero è uguale ad 1 l'indirizzo dell'istruzione ANI è conservato alla sommità dello Stack. Il Puntatore dello Stack è decrementato di 2. La prossima istruzione eseguita sarà quella con la label SUBR.

DAA – AGGIUSTA DECIMALI ACCUMULATORE

DAA
27

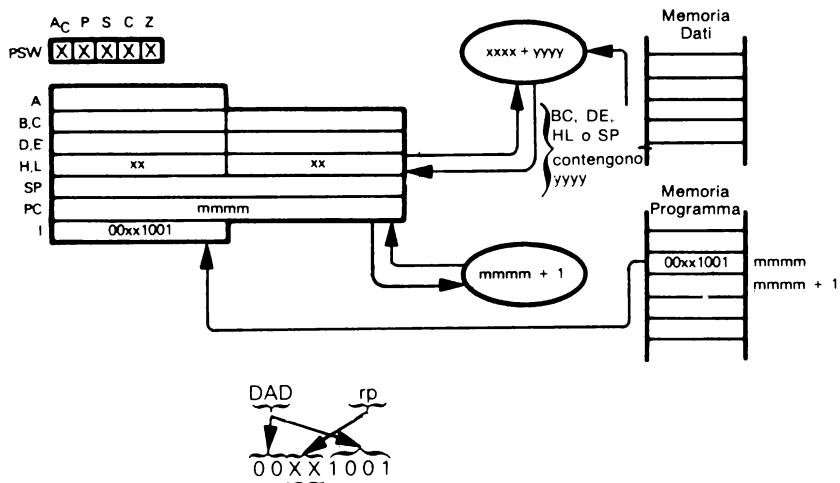
Converte i contenuti dell'Accumulatore nella loro forma decimale codificato binario. Questa istruzione potrebbe essere usata solo dopo l'addizione di due numeri BCD, cioè come si è visto ADD DAA oppure ADC DAA o SUB DAA o SBB DAA come istruzioni composte, aritmetiche decimali che operano in sorgente BCD per generare risposte BCD.

Si supponga che l'accumulatore contenga 39_{16} ed il registro B contenga 47_{16}. Dopo che le istruzioni:

```
ADD     B
DAA
```

sono state eseguite, l'Accumulatore conterrà 86_{16} e non 80_{16}.

L'istruzione DAA modifica tutti i flag di stato, ma solo Carry è significativo.

DAD — SOMMA UNA COPPIA DI REGISTRI AD H ED L

DAD rp

0 0 X X 1 0 0 1

00 per rp = B, rappresentante B, C
01 per rp = D, rappresentante D, E
10 per rp = H, rappresentante H, L
11 per rp = SP, rappresentante il Puntatore dello Stack

Somma il valore a 16 bit dalle coppie BC, DE o HL oppure il Puntatore dello Stack alla coppia di registri HL.

Si supponga che H, L contenga $034A_{16}$ e B, C contenga $214C_{16}$. Dopo che l'istruzione:

DAD B

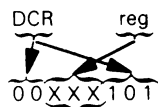
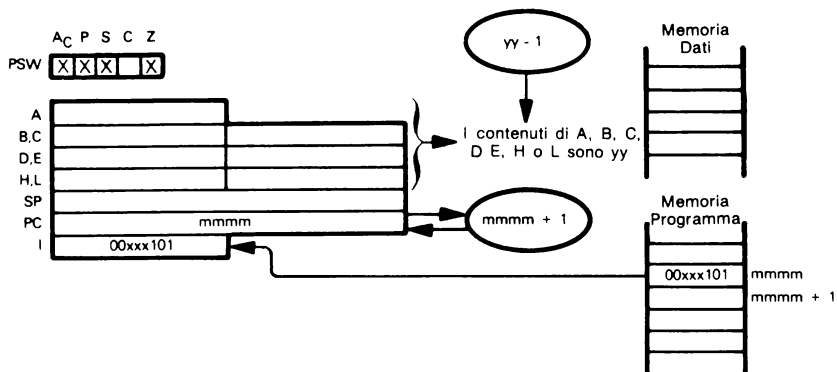
è stata eseguita, la coppia di registri H, C conterrà 2496_{16}:

```
034A = 0000001101001010
214C = 0010000101001100
       ----------------
       0010010010010110
```

Non c'è carry così C è ripristinato a 0

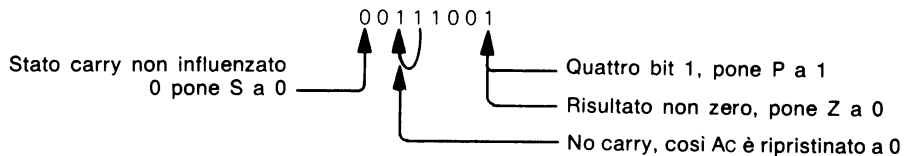
Nessun altro stato è influenzato

L'istruzione DAD è una delle più usate nel set di istruzione dell'8080, per applicazioni di programmazione tradizionale. Questa istruzione fornisce l'equivalente di uno spostamento a sinistra di 16 bit.

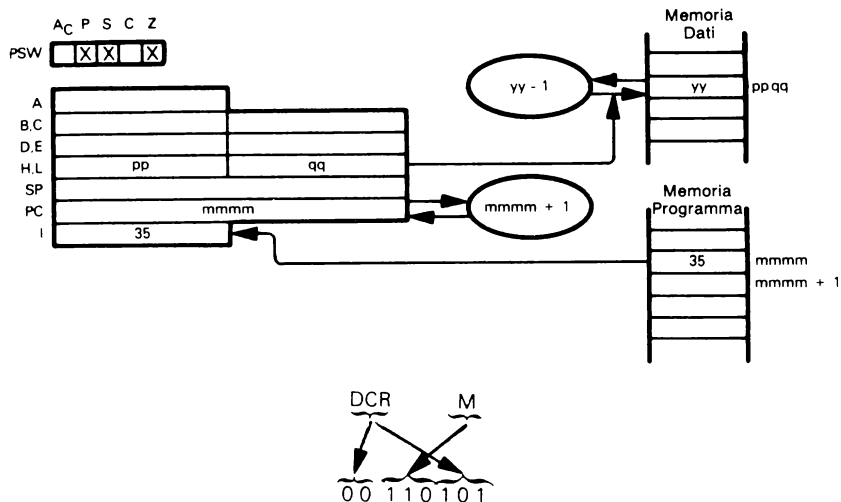
DCR — DECREMENTA I CONTENUTI DI REGISTRO DI MEMORIA

Sottrae 1 dai contenuti del registro specificato.

Si supponga che il registro C contenga $3A_{16}$. Dopo che l'istruzione:

DCR C

è stata eseguita, il registro C conterrà 39_{16}:

Anche i contenuti del byte della memoria di lettura/scrittura possono essere decrementati:

DCR M

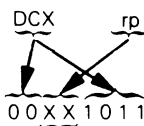
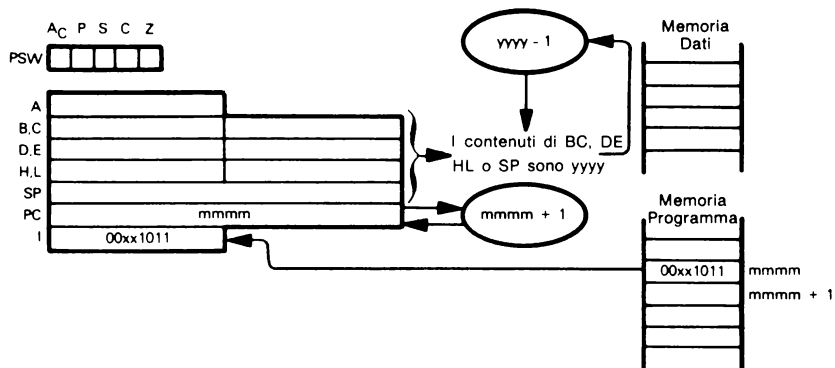
00 110101

Si supponga che HL contenga 3714_{16}. L'esecuzione dell'istruzione:

DCR M

sottrae 1 dai contenenti del byte di memoria con indirizzo 3714_{16}. I flag di stato sono modificati come descritto per l'istruzione DCR C.

L'istruzione è usata in cicli di istruzione iterativi che impiegano un contatore dal valore minore uguale a 256. Una forma tipica di ciclo è la seguente:

```
MVI     reg.    dati    ;CARICA IL VALORE INIZIALE DEL CONTATORE
LOOP    —               ;PRIMA ISTRUZIONE DEL CICLO
        —
        —
        —
        DCR     reg     ;DECREMENTA IL CONTATORE
        JNZ     LOOP    ;RITORNA SE NON ZERO
```

DCX — DECREMENTA UNA COPPIA DI REGISTRI

00 per rp = B, rappresentante B, C
01 per rp = D, rappresentante D, E
10 per rp = H, rappresentante H, L
11 per rp = SP, rappresentante il Puntatore dello Stack

Sottrae 1 dal valore a 16 bit contenuto nella coppia di registri specificata.

Si supponga che il Contatore dello Stack contenga $2F7A_{16}$. Dopo che l'istruzione:

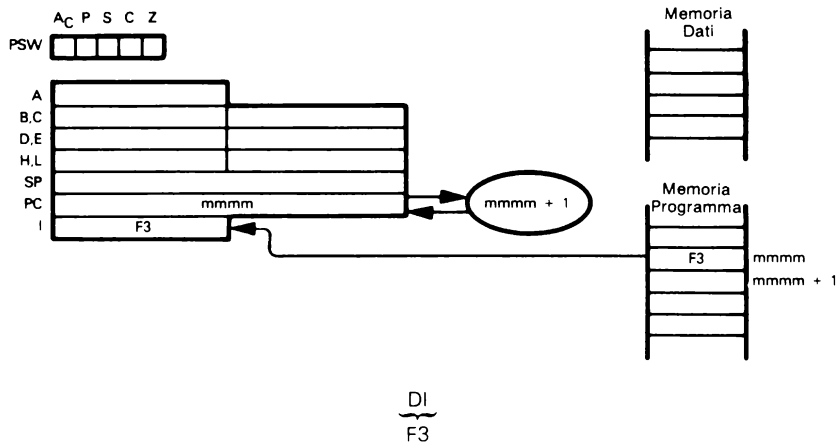
DCX SP

è stata eseguita, il contatore dello Stack conterrà $2F79_{16}$.

L'istruzione DCX non modifica nessun flag di stato e questo è un difetto nel set di istruzioni dell'8080. Mentre l'istruzione DCR è usata nei cicli iterativi di istruzioni che usano un contatore con un valore minore o uguale a 256, l'istruzione DCX deve essere usata se il valore del contatore è maggiore di 256. Poichè l'istruzione DCX non influenza i flag di stato, deve essere semplicemente aggiunta al testo un'altra istruzione per il risultato zero. Una forma tipica di ciclo è la seguente:

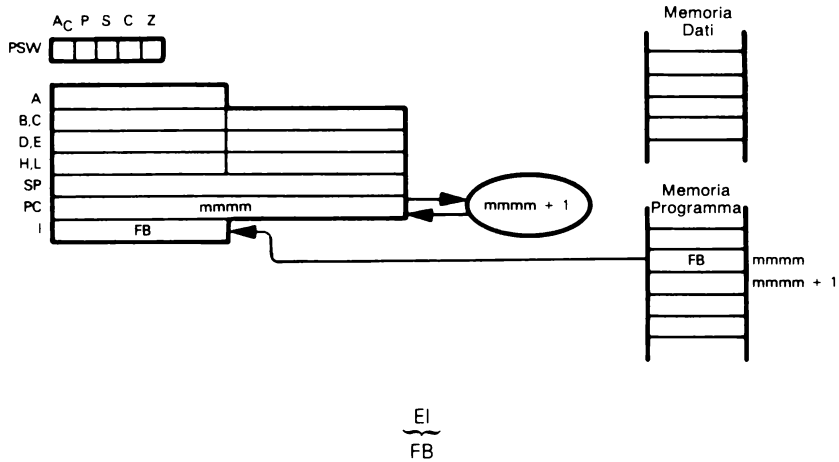
```
        LXI     D, dati 16  ;CARICA IL VALORE INIZIALE A 16 BIT NEL CONTATORE
LOOP    —       —           ;PRIMA ISTRUZIONE DEL CICLO
        —       —
        —       —
        —       —
        DCX     D           ;DECREMENTA IL CONTATORE
        MOV     A,D         ;PER IL TEST PER ZERO, MOVE D AD A
        ORA     E           ;QUINDI OR DI A CON E
        JNZ     LOOP        ;RITORNA SE NON ZERO
```

DI — DISABILITA INTERRUPT

DI
F3

Dopo che è stata eseguita questa istruzione il segnale INTE è basso all'uscita della CPU dell'8080 e non verrà accettata nessuna richiesta di interrupt. Nessun registro o flag è influenzato.

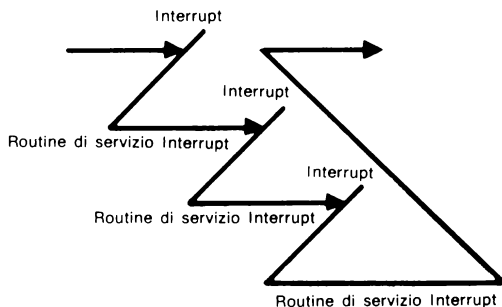
Si ricordi che quando un interrupt è accettato, gli interrupt sono automaticamente disabilitati.

EI — ABILITA INTERRUPT

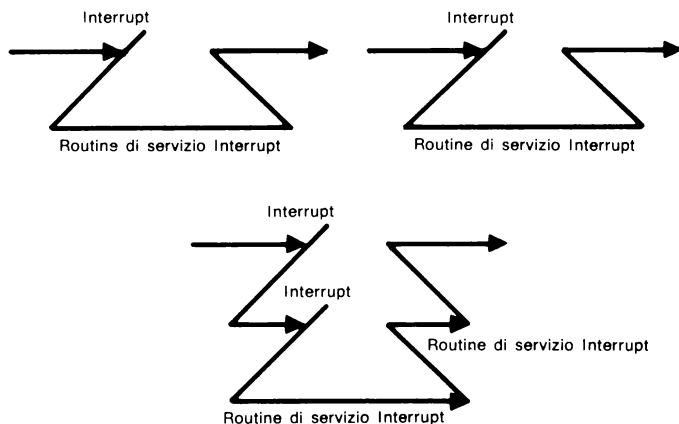
EI
FB

Quando è eseguita questa istruzione, gli interrupt sono abilitati a partire dall'istante in cui è terminata l'istruzione che si stava eseguendo.
La maggior parte del routine di servizio interrupt terminano con le due istruzioni:

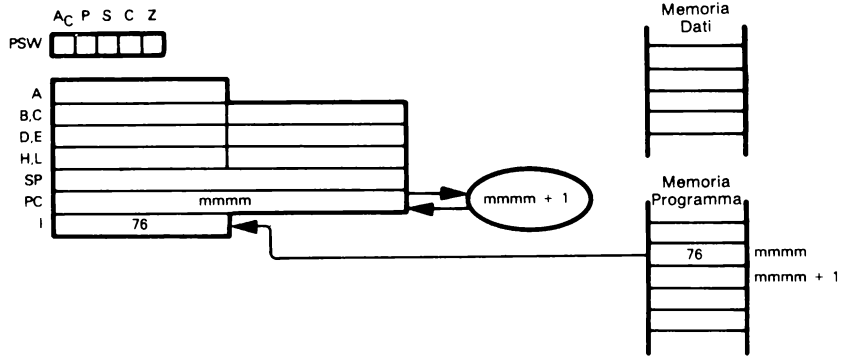
```
EI      ;ABILITA GLI INTERRUPT
RET     ;RITORNA AL PROGRAMMA INTERROTTO
```

Se gli interrupt sono eseguiti sequenzialmente, allora per l'intera durata della routine di servizio interrupt, tutti gli interrupt sono disabilitati – che significa che in una applicazione a multi-interrupt c'è una significativa possibilità per uno o più interrupt di essere sospeso fino a quando una qualsiasi routine di servizio interrupt completa l'esecuzione.

Se gli interrupt erano ammessi non appena l'istruzione EI è stata eseguita, allora l'istruzione di ritorno non sarà eseguita. In queste circostanze gli interrupt potrebbero impilarsi uno sull'altro e consumare dello Stack di memoria non necessario. Questo può essere illustrato come segue:

Inibendo gli interrupt per più istruzioni l'esecuzione di EI, la CPU dell'8080 assicura che l'istruzione RET è eseguita con la sequenza:

```
–
–
–
EI        ;ABILITA GLI INTERRUPT
RET       ;RITORNO DA INTERRUPT
```

È abbastanza comune per gli interrupt non essere abilitati mentre si sta eseguendo una routine di servizio interrupt. Gli interrupt sono processati in serie:

Se gli interrupt sono processati in serie l'arbitraggio di priorità sarà applicata soltanto durante i processi di ammissione.

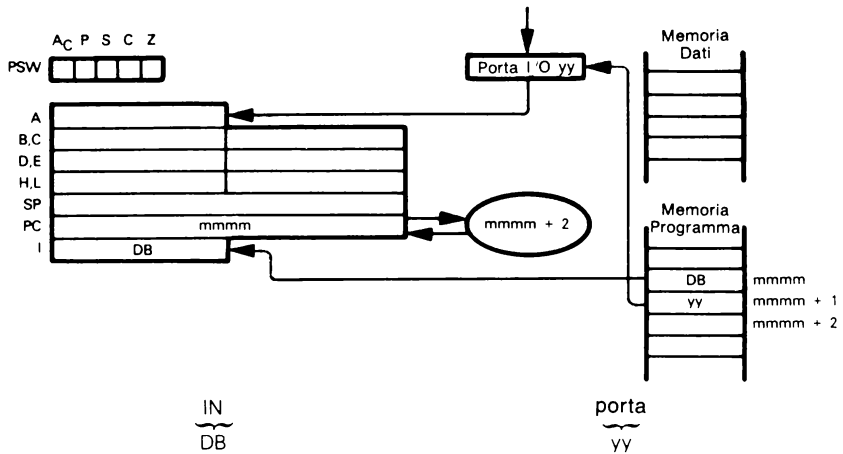
HLT — ARRESTO (HALT)

$$\underbrace{\text{HLT}}_{76}$$

Quando viene eseguita l'istruzione HLT cessa l'esecuzione del programma. Per fare ripartire l'esecuzione si richiede un interrupt od un reset. Nessun registro o stato viene influenzato.

NOTA: Gli interrupt non sono abilitati da una istruzione EI precedente l'istruzione HALT, la CPU dell'8080 non può uscire dallo stato HALT se non mediante attivazione tramite ripristino hardware.

IN — INGRESSO ALL'ACCUMULATORE

IN   porta

DB   yy

Carica un byte di dati nell'Accumulatore della porta I/O identificata dal secondo byte dell'istruzione in codice oggetto IN.

Si supponga che il Buffer della Porta I/O $1A_{16}$ contenga 36_{16}. Dopo che l'istruzione:

IN 1AH

è stata eseguita, l'Accumulatore conterrà 36_{16}. L'istruzione IN non influenza nessuno stato.

L'uso dell'istruzione IN dipende molto dall'hardware. La validità di indirizzi per porte I/O è determinata dal modo in cui è stata realizzata la logica I/O. È anche possibile progettare un sistema a microcomputer impiegante istruzione di reference della memoria con specifici indirizzi di memoria.

INR — INCREMENTA REGISTRO O CONTENUTI DI MEMORIA

AC P S C Z
PSW X X X X

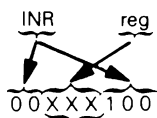
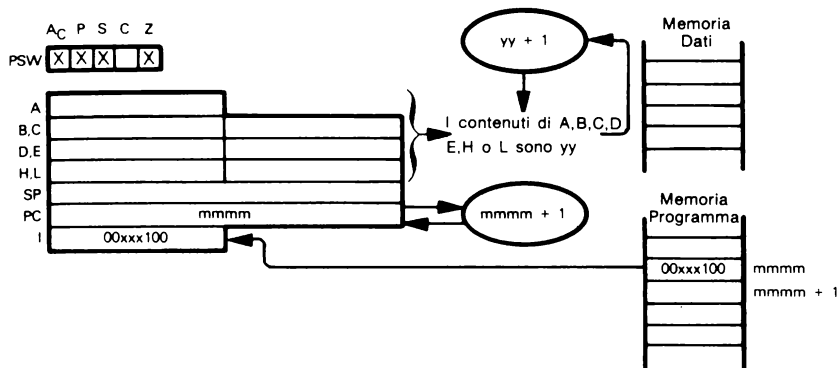
A
B,C
D,E
H,L
SP
PC mmmm
I 00xxx100

yy + 1
I contenuti di A,B,C,D
E,H o L sono yy
mmmm + 1

Memoria Dati

Memoria Programma
00xxx100 mmmm
mmmm + 1

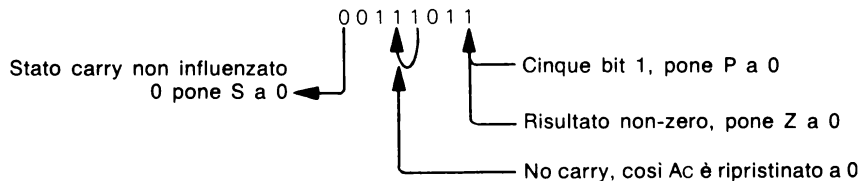
INR reg

0 0 X X X 1 0 0

000 per reg = B
001 per reg = C
010 per reg = D
011 per reg = E
100 per reg = H
101 per reg = L
111 per reg = A

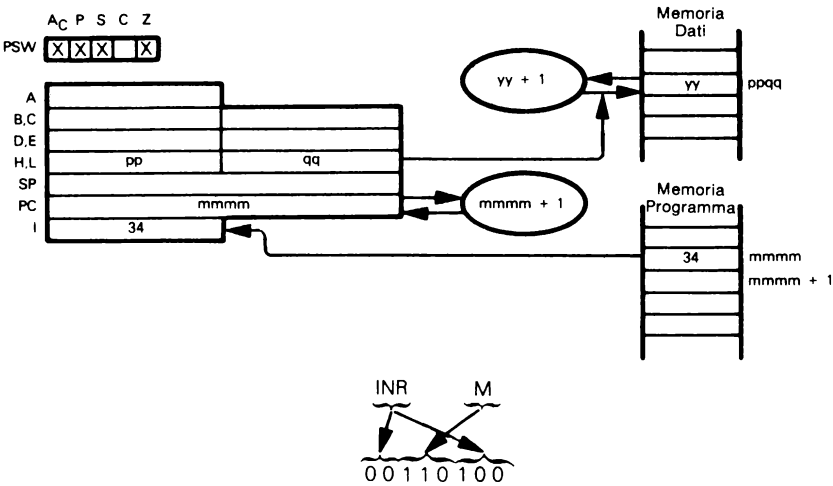
Aggiunge 1 di contenuti del registro specificato.

Si supponga che il registro C contenga $3A_{16}$. Dopo che l'istruzione:

INR C

è stata eseguita, il registro conterrà $3B_{16}$:

Anche i contenuti del byte della memoria di lettura/scrittura possono essere incrementati:

INR M

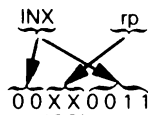
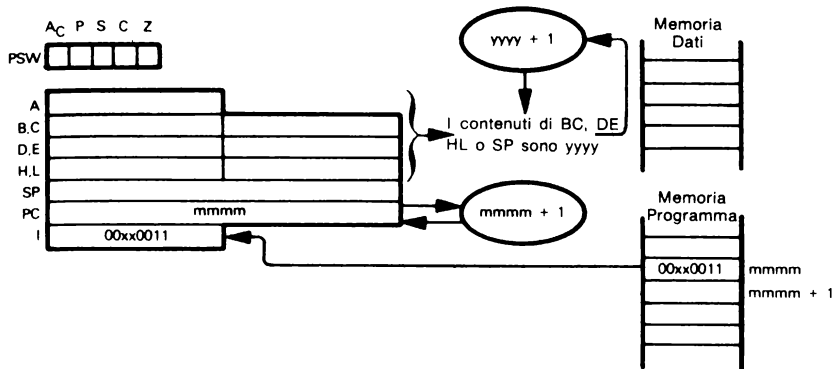
00110100

Si supponga che HL contenga 3714_{16}. Allora l'esecuzione dell'istruzione:

INR M

aggiunge 1 ai contenuti del byte di memoria con l'indirizzo 3714_{16}. I flag di stato sono modificati come descritto per l'istruzione INR C.

L'istruzione INR è usata nei cicli iterativi di istruzioni che usano un contatore di valore minore uguale a 256. Una forma tipica di ciclo è la seguente:

```
       MVI   reg. dati  ;CARICA IL COMMENTO DEL VALORE INIZIALE
                        ;DEL CONTATORE
LOOP   –     –          ;PRIMA ISTRUZIONE DEL CICLO
       –     –
       –     –
       INR   reg.       ;DECREMENTA IL CONTATORE
       JNZ   LOOP       ;RITORNA SE NON ZERO
```

INX — INCREMENTA COPPIA DI REGISTRI

00 per rp = B, rappresentante BC
01 per rp = D, rappresentante DE
10 per rp = H, rappresentante HL
11 per rp = SP, rappresentante il Puntatore dello Stack

Somma 1 valore a 16 bit contenuto nella coppia di registri specificata.

Supponendo che i registri D ed E contengano $2F7A_{16}$. Dopo che l'istruzione:

INX D

è stata eseguita, i registri D ed E conterranno $2F7B_{16}$.

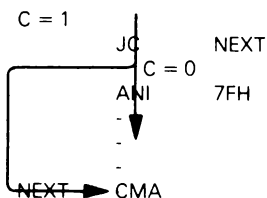
L'istruzione INX non modifica nessun flag di stato, questo è un difetto nel set di istruzioni dell'8080. Mentre l'istruzione DCR è usata nei cicli iterativi di istruzione che usano un contatore con valore minore uguale di 256, l'istruzione INX deve essere usata se il valore del contatore è maggiore di 256. Poichè l'istruzione INX non impone flag di stato, occorre semplicemente aggiungere altre istruzioni per il test di risultato zero. Una forma tipica di ciclo è la seguente:

```
        LXI    D, dati 16 ;CARICA IL COMPLEMENTO A 16 BIT DEL
                          ;VALORE INIZIALE DEL CONTATORE
LOOP    —      —          ;PRIMA ISTRUZIONE DEL CICLO
        —      —
        —      —
        —      —
        INX    D          ;INCREMENTA IL CONTATORE
        MOV    AD         ;PER IL TEST PER ZERO, MUOVI D AD A
        ORA    E          ;QUINDI OR DI A CON E
        JNZ    LOOP       ;RITORNA SE NON ZERO
```

JC — SALTA SE CARRY

JC	label
DA	ppqq

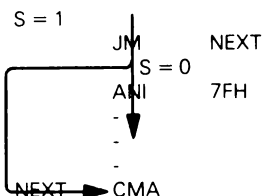
Questa istruzione è identica all'istruzione JMP solo che qui l'istruzione di Salto è eseguita solo se lo stato Carry è uguale ad 1; altrimenti viene eseguita l'istruzione sequenzialmente successiva.

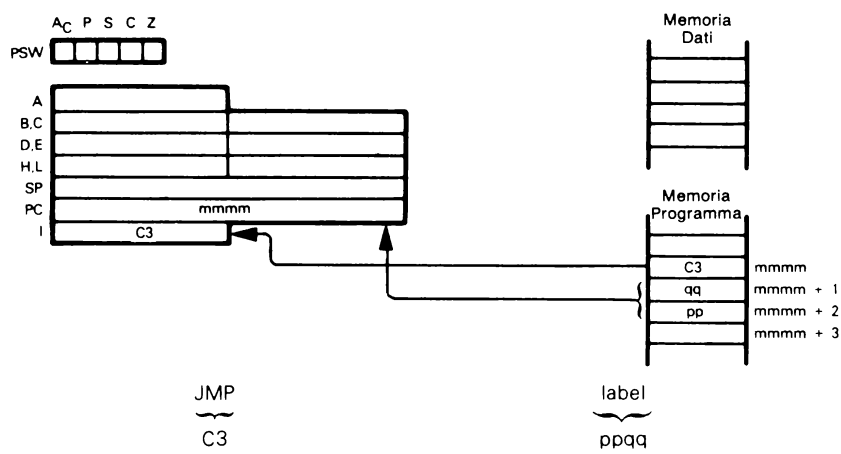
Si osservi la seguente sequenza di istruzioni:

Dopo l'istruzione JC viene eseguita la CMA, se lo stato Carry è uguale ad 1. L'istruzione ANI viene eseguita se lo stato Carry è uguale a 0.

JM — SALTA SE MENO

JM	label
FA	ppqq

Questa istruzione è identica alla JMP eccetto che il salto è eseguito solo se lo stato Sign è uguale ad 1; altrimenti verrà eseguita la successiva istruzione.

Si osservi la seguente sequenza di istruzioni:

Dopo l'istruzione JM viene eseguita l'istruzione CMA se lo stato Sign è uguale ad 1. L'istruzione ANI è eseguita se lo stato Sign è uguale a 0.

JMP – SALTA ALL'ISTRUZIONE IDENTIFICATA NELL'OPERANDO

Carica i contenuti del secondo e terzo byte dell'istruzione di Salto in codice oggetto nel Contatore di Programma; questo diventa l'indirizzo di memoria per la successiva istruzione da eseguire. Il precedente contenuto del Contatore di Programma viene perso.

Si osservi la seguente seguenza di istruzioni:

```
        JMP   NEXT
        ANI   7FH
        –
        –
        –
NEXT    CMA
```

Dopo l'istruzione JMP sarà eseguita l'istruzione CMA. L'istruzione ANI non verrà eseguita fino a che una qualche istruzione di salto della sequenza di istruzione non opera un salto a questa istruzione.

JNC – SALTA SE NO CARRY

JNC → D2   label → ppqq

Questa istruzione è identica all'istruzione JMP eccetto che il salto è eseguito solo lo stato Carry è uguale a 0; diversamente viene eseguita l'istruzione successiva.

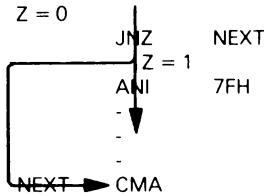
Si osservi la seguente sequenza di istruzioni:

Dopo l'istruzione JNC viene eseguita l'istruzione CMA se lo stato Carry è uguale a 0. L'istruzione ANI viene eseguita se lo stato Carry è uguale ad 1.

JNZ — SALTA SE NON ZERO

JNZ	label
C2	ppqq

Questa istruzione è identica alla JMP eccetto che il salto viene eseguito solo se lo stato Zero è uguale a 0; altrimenti viene eseguita l'istruzione successiva.

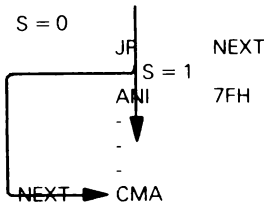
Si osservi la seguente sequenza di istruzione:

Dopo l'istruzione JNZ viene eseguita la CMA se lo stato Zero è uguale a 0.

L'istruzione ANI viene eseguita se lo stato Zero è uguale ad 1.

JP — SALTA SE POSITIVO

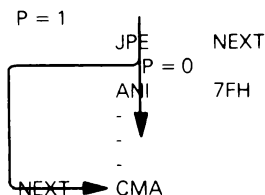
JP	label
F2	ppqq

Questa istruzione è identica alla JMP tranne il fatto che il salto viene eseguito solo se lo stato Sign vale 0; altrimenti viene eseguita l'istruzione successiva.

Si osservi la seguente sequenza di istruzione:

Dopo l'istruzione JP viene eseguita la CMA se lo stato Sign è guale a 0. L'istruzione ANI viene eseguita se lo Stato Sign è uguale ad 1.

JPE — SALTA SE PARITÀ PARI

Questa istruzione è identica alla JMP tranne il fatto che il salto viene eseguito solo se lo stato Parità è uguale ad 1; altrimenti viene eseguita l'istruzione successiva.

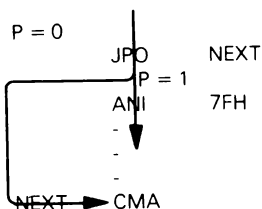
Si osservi la seguente sequenza di istruzione:

Dopo l'istruzione JPE viene eseguita la CMA se lo stato Parità è uguale ad 1.

L'istruzione ANI viene eseguita se lo stato parità è uguale a 0.

JPO — SALTA SE PARITÀ DISPARI

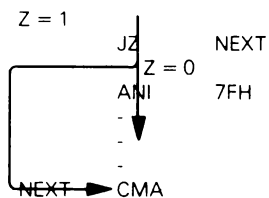
JPO — E2   label — ppqq

Questa istruzione è identica alla JMP tranne il fatto che il salto è eseguito solo se lo stato Parità vale 0; altrimenti viene eseguita l'istruzione successiva.

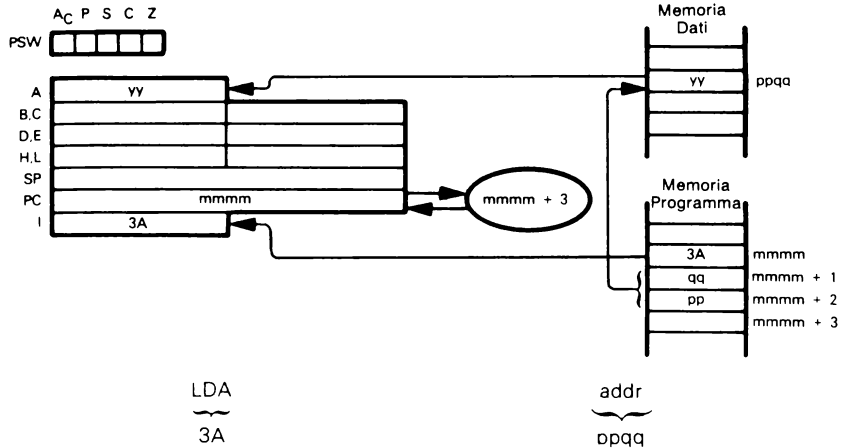
Si osservi la seguente sequenza di istruzione:

Dopo l'istruzione JPO viene eseguita la CMA se lo stato Parità è uguale a 0.
L'istruzione ANI viene eseguita se lo stato Parità è uguale ad 1.

JZ — SALTA SE ZERO

Questa istruzione è identica alla JMP tranne il fatto che il salto è eseguito solo se lo stato Zero è uguale ad 1; altrimenti viene eseguita la prossima istruzione.

Si osservi la seguente sequenza di istruzioni:

Dopo l'istruzione JZ viene eseguita la CMA se lo stato Zero è uguale ad 1.
L'istruzione ANI viene eseguita se lo stato Zero è uguale a 0.

LDA – CARICA L'ACCUMULATORE DALLA MEMORIA UTILIZZANDO L'INDIRIZZAMENTO DIRETTO

LDA — 3A   addr — ppqq

Carica nell'Accumulatore il contenuto del byte di memoria indirizzato direttamente dal secondo e terzo byte dell'istruzione LDA in codice oggetto.

Si supponga che il byte di memoria $084A_{16}$ contenga $3A_{16}$. Dopo che l'istruzione:

```
LABEL   EQU   084AH
        —
        —
        —
        LDA   LABEL
```

è stata eseguita l'Accumulatore conterrà $3A_{16}$.

Si ricordi che EQU è un Ordine Assembler e non una istruzione; esso impone l'uso del valore a 16 bit $084A_{16}$ dovunque appare LABEL.

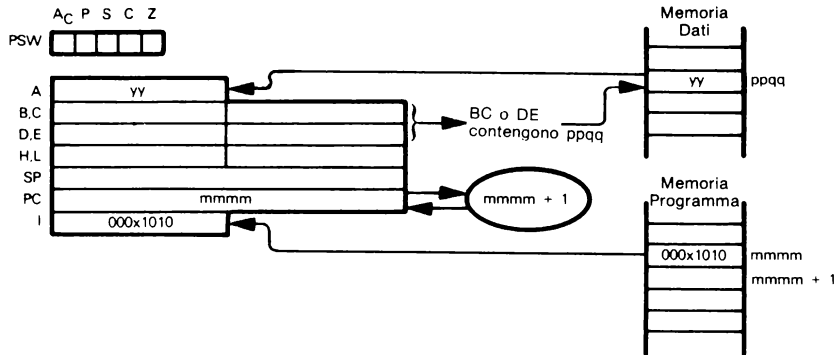
L'istruzione:

```
LDA   LABEL
```

è equivalente alle due istruzioni:

```
LXI   H,LABEL
MOV   A,M
```

L'istruzione LDA è preferita quando si sta caricando un singolo dato dalla memoria; essa usa una istruzione e tre byte in programma oggetto per fare quello che fa la combinazione LXI MOV in due istruzioni e quattro byte in programma oggetto. Inoltre la combinazione LXI MOV usa i registri H ed L; l'istruzione LDA invece no.

LDAX — CARICA L'ACCUMULATORE DALLA LOCAZIONE DI MEMORIA INDIRIZZATA MEDIANTE UNA COPPIA DI REGISTRI

Carica nell'Accumulatore i contenuti del byte di memoria indirizzato dalla coppia di registro BC o DE.

Si supponga che il registro B contenga 08_{16}, il registro C contenga $4A_{16}$ ed il byte di memoria $084A_{16}$ contenga $3A_{16}$. Dopo che l'istruzione:

```
LDAX    B
```

è stata eseguita, l'accumulatore conterrà $3A_{16}$.

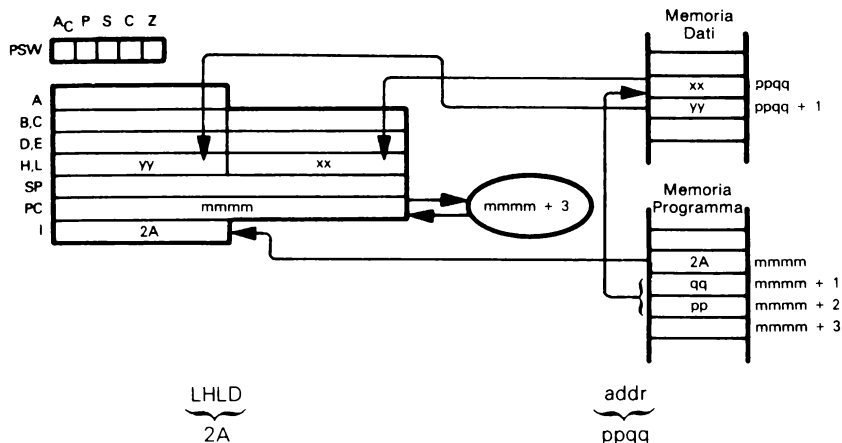
Si noti che non esiste l'istruzione LDAX H poichè è identica all'istruzione MOV A,M.

Le istruzioni LDAX ed LXI saranno usate normalmente assieme, poichè l'istruzione LXI carica un indirizzo a 16 bit nei registri BC o DE come segue:

```
LXI     B,084AH
LDAX    B
```

Si noti che l'istruzione LDAX caricherà dati soltanto nell'Accumulatore mentre l'istruzione MOV caricherà dati in qualsiasi registro.

LHLD — CARICA DIRETTAMENTE I REGISTRI H ED L

LHLD → 2A   addr → ppqq

Il secondo e terzo byte in codice oggetto forniscono l'indirizzo di memoria di un byte di dati, il cui contenuto sarà caricato nel registro L. Il contenuto del byte dati sequenzialmente successivo è caricato nel registro H.

Si supponga che il byte di memoria $084A_{16}$ contenga $3A_{16}$ ed il byte $084B_{16}$ contenga $2C_{16}$. Dopo che l'istruzione:

```
LABEL   EQU    084AH
        —
        —
        —
        LHLD   LABEL
```

è stata eseguita, il registro H conterrà $2C_{16}$ ed il registro L conterrà $3A_{16}$.

Si ricordi che EQU è un ordine assembler e non una istruzione; esso dice all'Assemblatore di usare il valore a 16 bit $084A_{16}$ dove si trova LABEL.

L'istruzione LHLD è una versione ad indirizzamento diretto della LXI H,DATI.

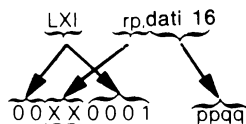
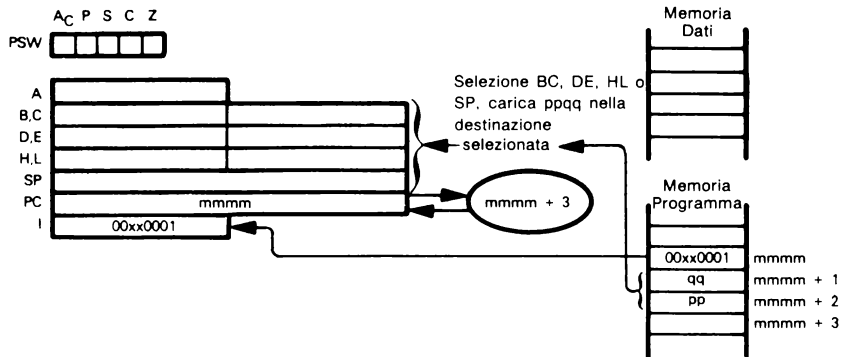
Per esempio l'istruzione:

```
        LXI    H,2C3AH
```

caricherà $2C_{16}$ nel registro H e $3A_{16}$ nel registro L.

Per dati a 16 bit non soggetti a variazioni, si usi LXI A, DATI al posto di LHLD ADDR.

Si ricordi, se IND indirizza direttamente un byte della memoria di lettura/scrittura, si può cambiare il valore che sarà caricato nei registri H ed L mediante una istruzione LHLD. Per fare questo si scriva semplicemente il nuovo valore in ADDR ed ADDR + 1.

LXI — CARICA UN VALORE A 16 BIT, IMMEDIATO, IN UNA COPPIA DI REGISTRI

00 se rp = B, selezionando i registri B e C
01 se rp = D, selezionando i registri D E
10 se rp = H, selezionando i registri H L
11 se rp = SP selezionando il Puntatore dello Stack

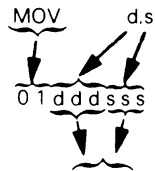
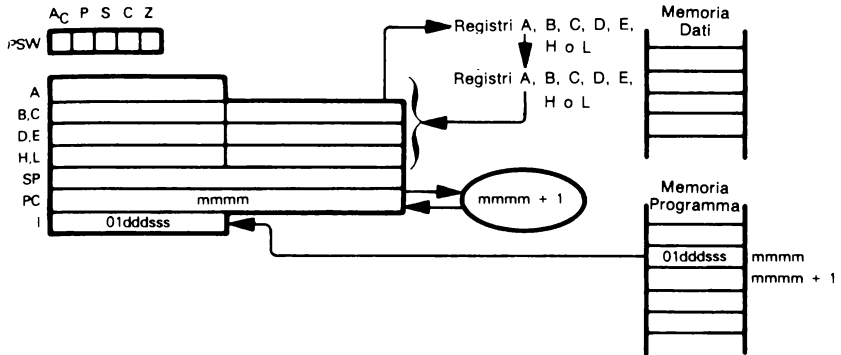
Carica nella coppia di registri selezionati i contenuti del secondo e terzo byte in codice oggetto.

Dopo che l'istruzione:

```
LXI     SP,217AH
```

è stata eseguita il Puntatore dello Stack conterrà $217A_{16}$. LXI è l'istruzione più spesso usata per il caricamento di indirizzi in una coppia di registri.

MOV — MUOVI DATI

Questa istruzione assume due forme. La prima opera il trasferimento dei contenuti di un registro in un altro.

Muove i contenuti di qualsiasi registro in qualunque altro. Per esempio:

MOV A, B

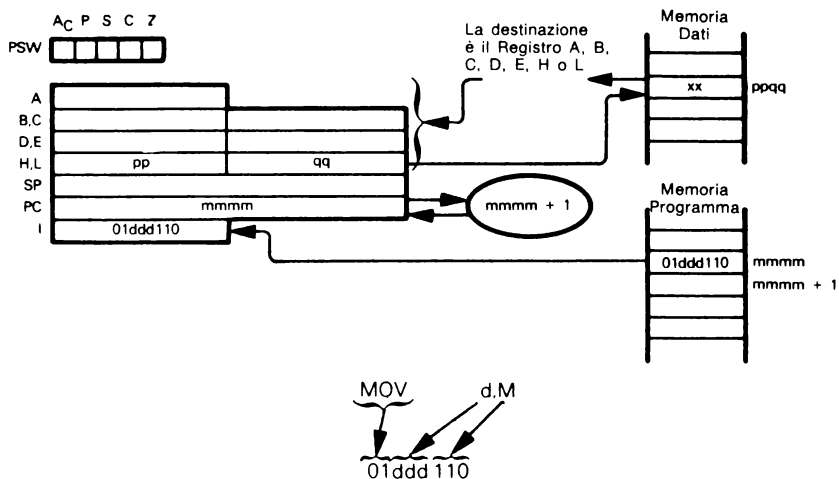
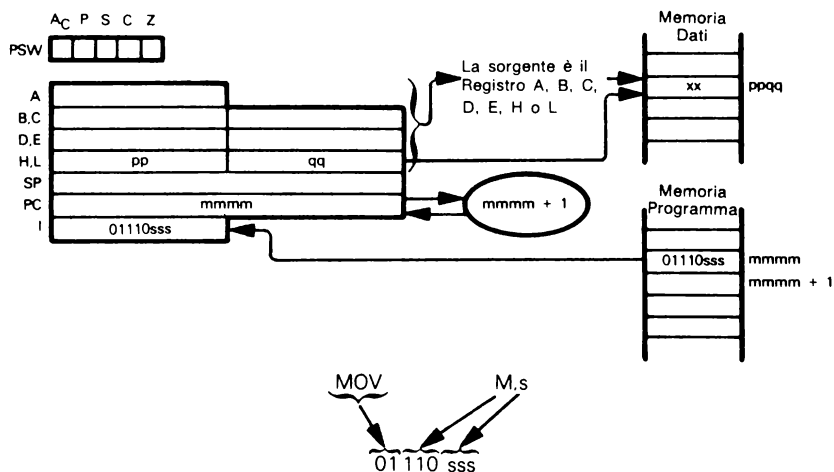
muove i contenuti del registro B nell'Accumulatore.

MOV L,D

muove i contenuti del registro D nel registro L.

MOV C,C

non succede nulla in quanto il registro C compare come sorgente e destinazione.

La sorgente dei dati può anche essere un byte di memoria:

Oppure un byte di memoria può essere la destinazione dei dati:

In entrambi i casi ddd o sss è il registro sul quale viene operato il movimento dati.

Così l'istruzione:

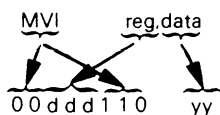
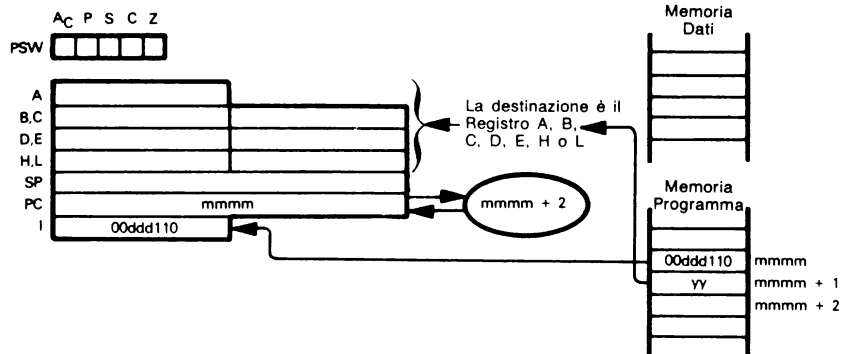
MOV M,A

muove il contenuto dell'Accumulatore nel byte di memoria di lettura/scrittura indirizzato mediante i registri H ed L. L'istruzione:

MOV L,M

muove i contenuti del byte di memoria indirizzato mediante i registri H ed L nel registro L.

L'istruzione di Movimento nelle sue varie forme è l'istruzione usata più frequentemente per l'8080.

MVI — MUOVI DATI IN MODO IMMEDIATO IN UN REGISTRO O IN MEMORIA

Muove i contenuti del secondo byte in codice oggetto ad uno dei registri.

Quando viene eseguita l'istruzione:

```
MVI     A,2AH
```

viene caricato $2A_{16}$ nell'Accumulatore. L'istruzione:

```
MVI     H,03H
```

carica 03_{16} nel registro H.

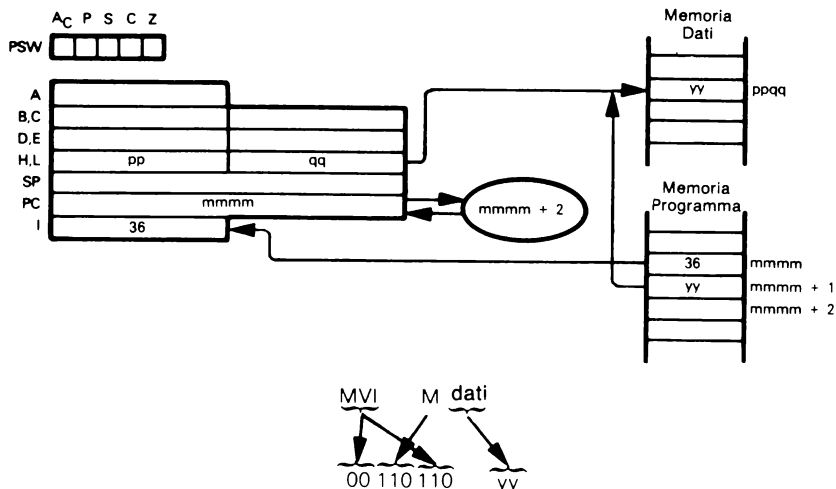
Le istruzioni di caricamento immediato dati su registro sono usate molto frequentemente nei programmi 8080.

Si noti che l'istruzione LXI è equivalente a due istruzioni MVI; per esempio:

```
LXI     H,032AH
```

è equivalente a:

```
MVI     H,03H
MVI     L,2AH
```

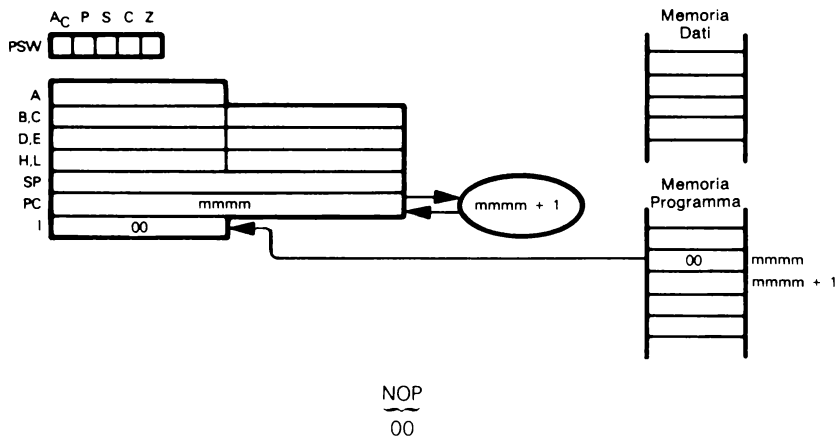
I dati possono anche essere caricati immediatamente in un byte della memoria di lettura/scrittura:

Si supponga che il registro H contenga 03_{16} ed il registro L contenga $2A_{16}$; quando viene eseguita l'istruzione:

MVI M,2CH

$2C_{16}$ sarà caricato nel byte di memoria $032A_{16}$.

L'istruzione di caricamento immediato in memoria (MVI M, dati) è usata molto meno dell'istruzione di caricamento immediato in Registro (MVI reg. dati).

NOP — NON OPERARE

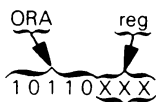
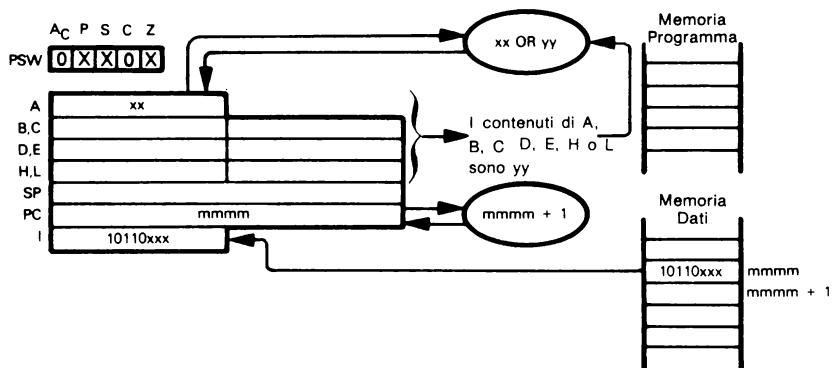
NOP
00

Quando viene eseguita questa istruzione non accade nulla; essa è presente per tre ragioni:

1) Quando è presente un errore di programma, che condurrebbe alla ricerca di un codice oggetto da una memoria non esistente, viene assegnato come codice oggetto 00. Questo è un buon artificio per assicurare l'eliminazione degli errori più comuni dei programmi.
2) L'istruzione NOP permette di ricavare una label da un byte in programma oggetto:

```
HERE    NOP
```

3) Per un buon accordo dei tempi di ritardo. Ogni istruzione NOP aggiunge quattro cicli di clock ad un ritardo.

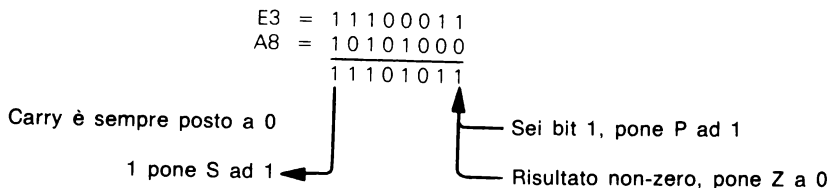
L'istruzione NOP non è molto pratica nè usata frequentemente.

ORA – OR DI UN REGISTRO O MEMORIA CON L'ACCUMULATORE

000 per reg. = B
001 per reg. = C
010 per reg. = D
011 per reg. = E
100 per reg. = H
101 per reg. = L
111 per reg. = A

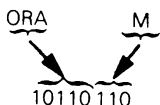
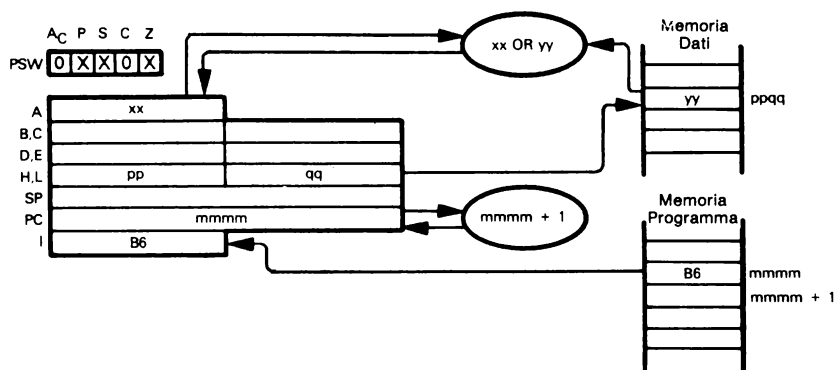
Opera l'OR dei contenuti dell'Accumulatore con qualunque altro registro. Il risultato è memorizzato nell'Accumulatore.

Si supponga xx = $E3_{16}$, il registro E contenga $A8_{16}$. Dopo che l'istruzione:

ORA E

è stata eseguita, l'Accumulatore conterrà EB_{16}.

Anche i contenuti del byte di memoria possono essere usati per operare l'OR con l'Accumulatore:

ORA M

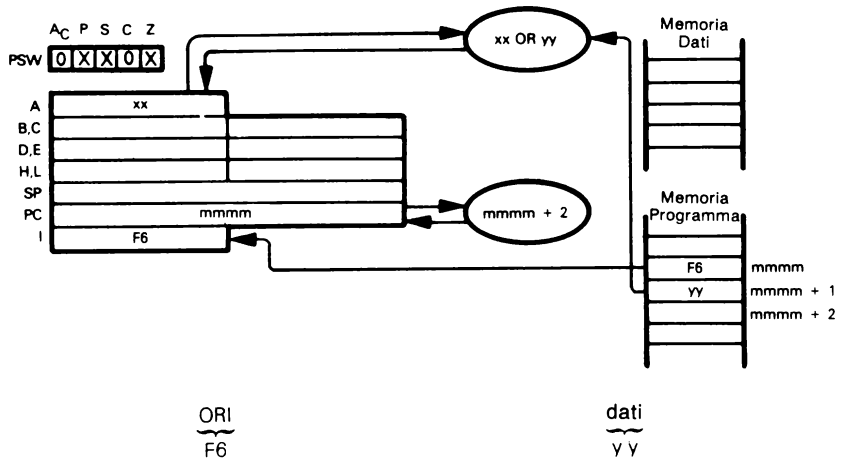
10110110

Se xx = $E3_{16}$ ed yy = $A8_{16}$ allora l'esecuzione dell'istruzione:

ORA M

genera lo stesso risultato dell'esecuzione dell'istruzione ORA E, appena descritta.

L'istruzione ORA non viene usata così frequentemente come l'istruzione di OR immediato (ORI).

Si noti che operando l'OR dell'Accumulatore con sè stesso (ORA A) permette di azzerare lo stato Carry; questa istruzione viene impiegata per porre gli stati dopo le istruzioni INX e DCX.

ORI — OR IMMEDIATO CON L'ACCUMULATORE

Opera l'OR dell'Accumulatore con i contenuti del secondo byte di istruzione in codice oggetto. Si supponga xx = $3A_{16}$. Dopo che è stata eseguita l'istruzione:

ORI 7CH

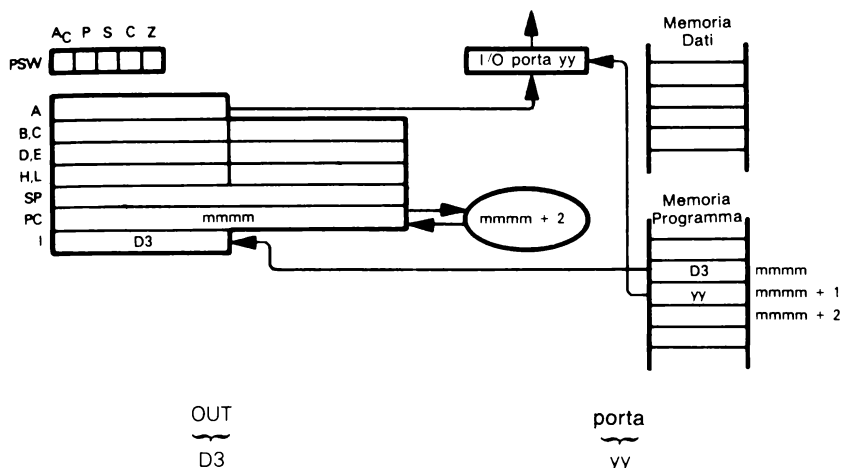
l'Accumulatore conterrà $7E_{16}$:

Questa è una istruzione logica di routine; viene spesso usata per commutare «on» dei bit. Per esempio l'istruzione

ORI 80H

porrà incondizionatamente ad 1 il bit di ordine più elevato dell'Accumulatore.

OUT — USCITA DALL'ACCUMULATORE

OUT	porta
D3	yy

Preleva i contenuti dall'Accumulatore e li fornisce alla porta I/O identifica dal secondo byte in codice oggetto dell'istruzione OUT.

Si supponga che l'Accumulatore contenga 36_{16}. Dopo che l'istruzione:

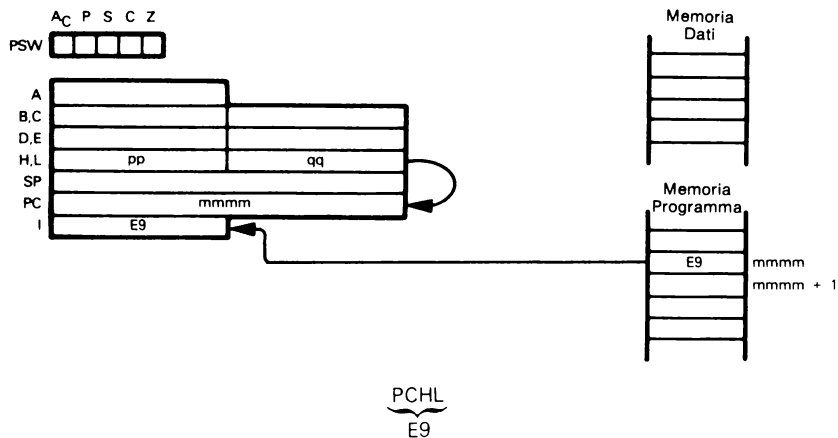
OUT 1AH

è stata eseguita, 36_{16} sarà stato trasferito nel buffer della porta I/O $1A_{16}$.

L'istruzione OUT non influenza nessuno stato.

L'uso dell'istruzione OUT dipende molto dall'hardware. Validi indirizzi della porta I/O sono determinati dal modo in cui è stata realizzata la logica I/O. È anche possibile progettare un sistema a microcomputer che acceda alla logica esterna mediante istruzioni di reference della memoria con specifici indirizzi di memoria.

L'istruzione OUT è spesso impiegata particolarmente per il controllo della logica esterna alla CPU del microcomputer.

PCHL – SALTA ALL'INDIRIZZO SPECIFICATO MEDIANTE HL

PCHL
E9

I contenuti dei registri H ed L vengono inviati al Contatore di Programma quindi viene eseguito un salto ad indirizzamento implicito.

La sequenza di istruzioni:

```
LXI     H,ADDR
PCHL
```

ha esattamente lo stesso effetto finale della singola istruzione:

```
JMP     ADDR
```

Entrambe specificano che la istruzione da eseguire è quella con la label ADDR.

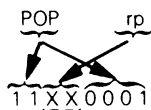
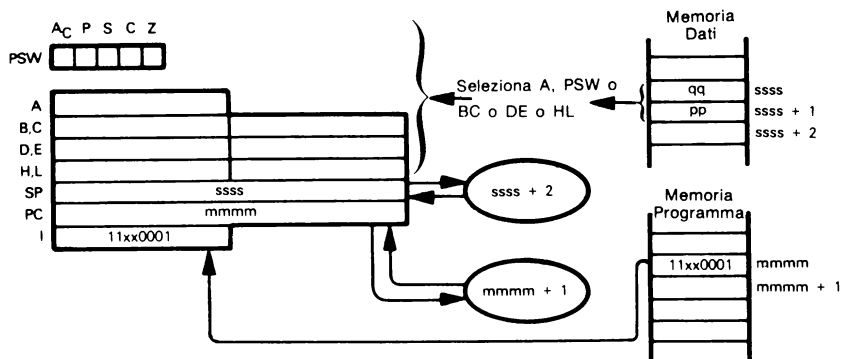
L'istruzione PCHL è comoda quando si vuole incrementare un indirizzo di ritorno di una subroutine avente ritorni multipli.

Si consideri la seguente chiamata della subroutine SUB:

```
CALL    SUB     ;CHIAMA SUBROUTINE
JMP     ERR     ;RITORNO ERRATO
                ;RITORNO ESATTO
```

Usando RET per ritornare dalla Sub il ritorno potrebbe essere operato dall'esecuzione della JUP ERR; perciò se la SUB viene eseguita senza rivelare condizioni di errore il ritorno avviene come segue:

```
POP     H       ;PONI L'INDIRIZZO DI RITORNO IN HL
INX     H       ;SOMMA 3 ALL'INDIRIZZO DI RITORNO
INX     H
INX     H
PCHL            ;RITORNO
```

POP — LEGGI DALLA SOMMITÀ DELLO STACK

00 se rp è B, selezionando i registri B e C
01 se rp è D, selezionando i registri D ed E
10 se rp è H, selezionando i registri H ed L
11 se rp è PSW, selezionando l'Accumulatore ed i flag di stato come una unità a 16 bit

Pone i due byte della sommità dello stack nella coppia di registri specificata.

Si supponga qq $= 03_{16}$ e pp $= 2A_{16}$. L'esecuzione dell'istruzione:

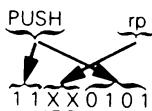
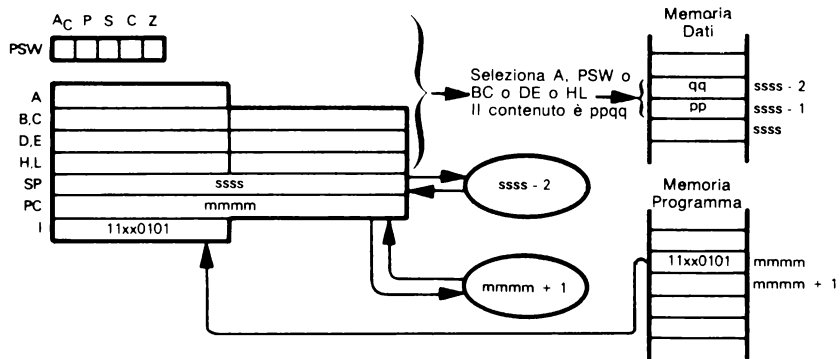
POP H

carica 03_{16} nel registro L e $2A_{16}$ nel registro H. L'esecuzione dell'istruzione:

POP PSW

carica 03_{16} nei flag di stato e $2A_{16}$ nell'Accumulatore. Così lo stato C sarà posto ad 1 e gli altri saranno azzerati.

L'istruzione POP è usata essenzialmente per rimemorizzare i contenuti di registri e stati che erano stati conservati nello stack, per esempio, mentre si stava eseguendo un interrupt.

PUSH — SCRIVI NELLA SOMMITÀ DELLO STACK

00 se rp è B, selezionando i registri B e C
01 se rp è D, selezionando i registri D ed E
10 se rp è H, selezionando i registri H ed L
11 se rp è PSW, selezionando l'Accumulatore ed i flag di stato come unità a 16 bit

Manda i contenuti della coppia di registri specificata alla sommità dello stack.

Si supponga che il registro H contenga 03_{16} ed L contenga $2A_{16}$.

L'esecuzione dell'istruzione:

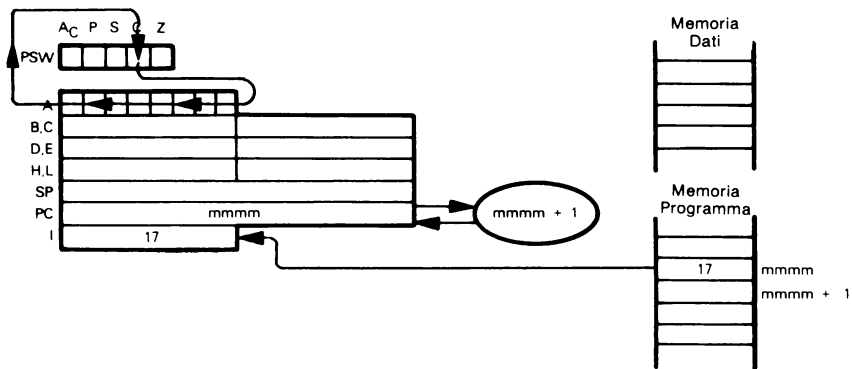
PUSH H

carica 03_{16} e poi $2A_{16}$ alla sommità dello Stack. L'esecuzione dell'istruzione:

PUSH PSW

carica l'Accumulatore e poi i flag di stato alla sommità dello Stack.

L'istruzione PUSH è usata essenzialmente per conservare i contenuti di registri e stati, per esempio, prima dell'esecuzione di un interrupt.

RAL — RUOTA L'ACCUMULATORE A SINISTRA ATTRAVERSO IL CARRY

$$\underbrace{\text{RAL}}_{17}$$

Ruota i contenuti dell'Accumulatore a sinistra di 1 bit attraverso lo stato Carry.

Si supponga che l'Accumulatore contenga $7A_{16}$ e lo stato Carry valga 1. Eseguita l'istruzione:

RAL

l'Accumulatore conterrà $F5_{16}$ e lo stato Carry sarà 0:

Accumulatore	C	→	Accumulatore	C
01111010	1		11110101	0

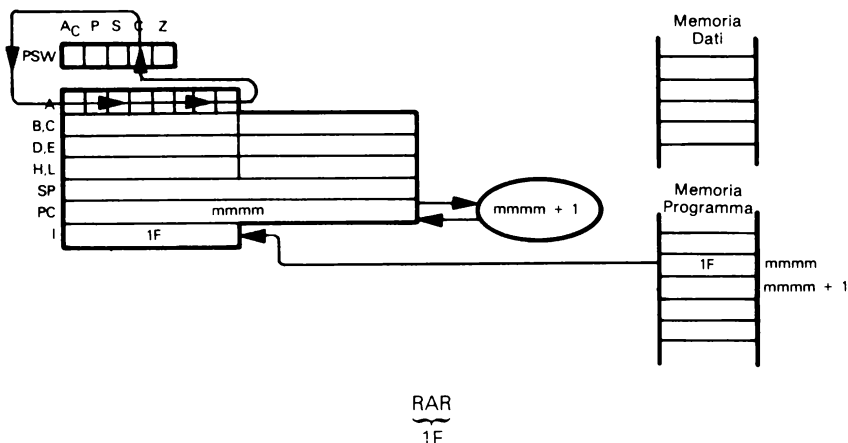
L'istruzione RAL è usata spesso per spostamenti multibyte a sinistra come descritto in An Introduction To Microcomputers — Volume 1 — Lo stato Carry è azzerato prima di eseguire il primo spostamento a sinistra; successivamente si trasferiscono i bit di ordine elevato byte nel successivo byte dei bit di ordine basso. Questa è una sequenza di istruzioni che sposta i contenuti di quattro byte di memoria a sinistra di un bit:

```
        LXI    H,DATI   ;CARICA L'INDIRIZZO DEL BYTE DATI DI BASSO ORDINE
        ANA    A        ;AZZERA INIZIALMENTE CARRY
        MVI    B,3      ;USA IL REGISTRO B COME CONTATORE
LOOP    MOV    A,M      ;CARICA IL BYTE DATI NELL'ACCUMULATORE
        RAL             ;RUOTA A SINISTRA
        MOV    M,A      ;RIMEMORIZZA IL RISULTATO
        INX    H        ;INCREMENTA L'INDIRIZZO IN HL
        DCR    B        ;DECREMENTA IL CONTATORE
        JNZ    LOOP     ;RITORNA PER PROSSIMO BYTE SE CE N'È UNO
```

CONDIZIONI DI STATO

Si noti che è stata data una particolare attenzione su come vengono influenzati gli stati dalle singole istruzioni. RAL influenza il solo Carry, INX e DCR gli stati Zero, Segno, Parità ma non il Carry che è perciò conservato nel passaggio da una esecuzione RAL alla successiva.

RAR — RUOTA L'ACCUMULATORE A DESTRA ATTRAVERSO IL CARRY

RAR
⏟
1F

Ruota a destra i contenuti dell'Accumulatore di un bit attraverso lo stato Carry.

Si supponga che l'Accumulatore contenga $7A_{16}$ e lo stato Carry sia 1.

Eseguita l'istruzione:

RAR

L'Accumulatore conterrà BD_{16} e lo stato Carry sarà 0:

Accumulatore	C		Accumulatore	C
01111010	1	→	10111101	0

L'istruzione RAR è usata spesso per eseguire spostamenti multibyte a destra, come descritto in An Introduction To Microcomputers. Volume I — Lo stato Carry è azzerato prima dell'esecuzione del primo spostamento a destra; successivamente si trasferiscono i bit di basso ordine di un byte nei bit di ordine elevato del byte successivo. Ecco una sequenza di istruzioni che sposta i contenuti di quattro byte di memoria a destra di un bit:

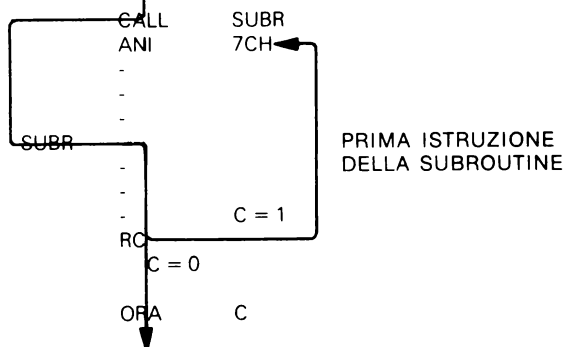
```
        LXI   H,DATI   ;CARICA L'INDIRIZZO DEL BYTE DATI DI BASSO ORDINE
        ANA   A        ;AZZERA INIZIALMENTE CARRY
        MVI   B,3      ;USA IL REGISTRO B COME CONTATORE
LOOP    MOV   A,M      ;CARICA IL BYTE DATI NELL'ACCUMULATORE
        RAR            ;RUOTA A DESTRA
        MOV   M,A      ;RIMEMORIZZA IL RISULTATO
        INX   H        ;INCREMENTA L'INDIRIZZO IN HL
        DCR   B        ;DECREMENTA IL CONTATORE
        JNZ   LOOP     ;RITORNA PER PROSSIMO BYTE SE CE N'È UNO
```

Si veda la descrizione dell'istruzione RAL per una discussione sugli stati.

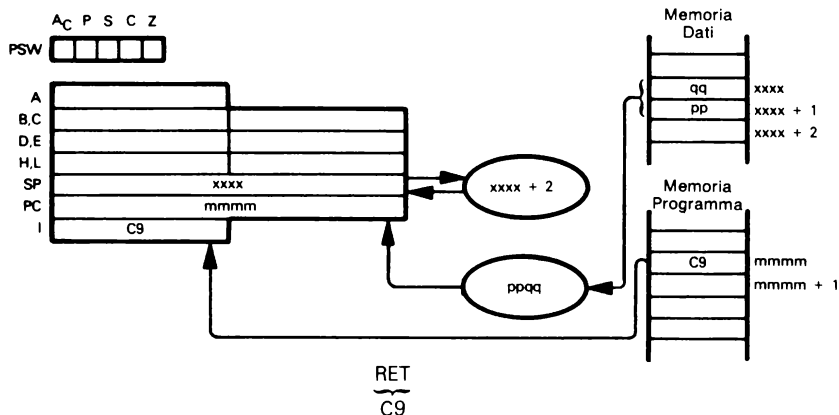
RC — RITORNA SE LO STATO CARRY È UGUALE AD 1

RC
D8

Questa istruzione è identica alla RET tranne il fatto che il ritorno non viene eseguito se lo stato Carry è diverso da 1, quando viene eseguita l'istruzione RC.

Si consideri la sequenza di istruzioni:

Eseguita l'istruzione RC, se lo stato Carry è uguale ad 1, l'esecuzione ritorna all'istruzione ANI che segue CALL. Se lo stato Carry è uguale a 0 viene eseguita l'istruzione ORA che è la successiva istruzione sequenziale.

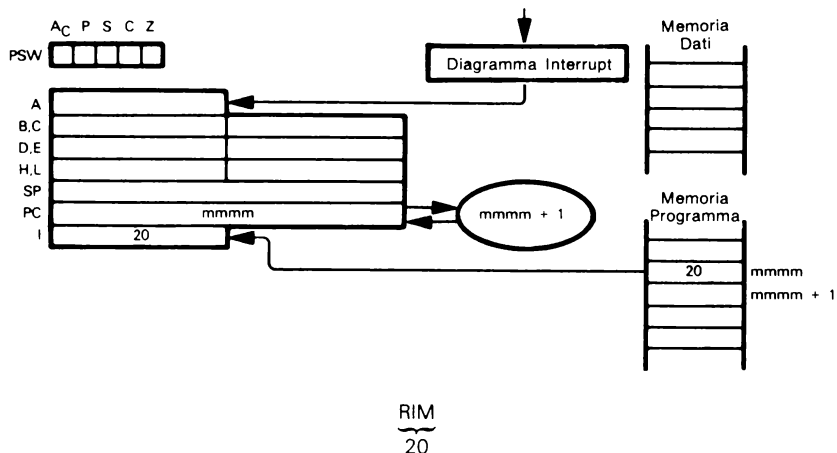
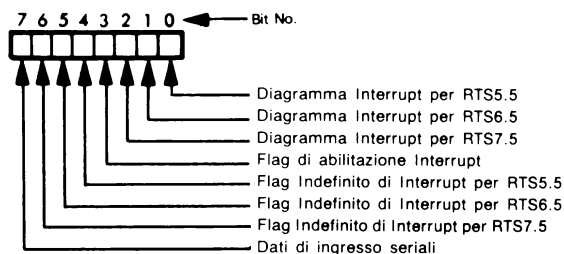
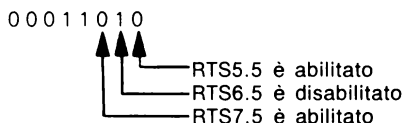
RET — RITORNA DA SUBROUTINE

RET
C9

Trasferisce i contenuti dei due byte alla sommità dello Stack al Contatore di Programma; questi due byte forniscono l'indirizzo della successiva istruzione da eseguire. Il precedente contenuto del Contatore di Programma va perso. Viene inoltre incrementato di 2 il Puntatore dello Stack per indirizzare la nuova sommità dello stack.

Ogni subroutine deve contenere almeno una istruzione di Ritorno (o Ritorno Condizionale); questa è l'ultima istruzione eseguita all'interno della subroutine ed origina l'esecuzione del ritorno al programma chiamante.

Per una dettagliata descrizione dell'esecuzione dell'istruzione RET, si veda il Capitolo 5.

RIM — LEGGI LA MASCHERA INTERRUPT

RIM
20

Carica i contenuti della maschera di ripristino da interrupt e la linea ad ingresso seriale nell'Accumulatore. I dati caricati nell'Accumulatore sono interpretati come segue:

I bit 0, 1 e 2 sono impiegati per le maschere di interrupt RST5.5, RST6.5 ed RST7.5. Se il bit corrispondente ad un particolare RST è ad 1, RST è disabilitato. Per esempio se una istruzione RIM fa in modo che $1A_{16}$ risulti caricato nell'Accumulatore, si ha:

Gli RST5.5, 6.5 e 7.5 operano in modo simile agli RST 0-7. Quando l'8085 rivela uno di questi RST, i suoi vettori risultano:

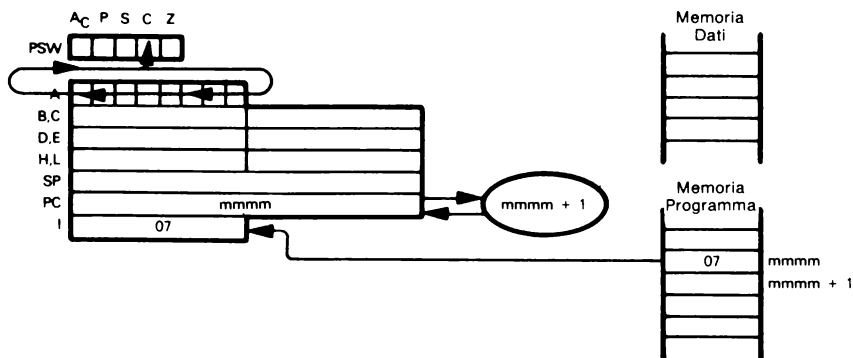
Vettori	a	**Locazione**
RST5.5		$002C_{16}$
RST6.5		0034_{16}
RST7.5		$003C_{16}$

Le altre informazioni del diagramma interrupt sono interpretate come segue:

Flag di abilitazione Interrupt – Se questo è ad 1, il sistema interrupt è abilitato se a 0 è disabilitato.

Flags di Interrupt in sospeso – Se questo flag è ad 1 c'è in sospeso la richiesta di interrupt dalla linea RST specificata. Se questo flag è a 0 non esiste richiesta di interrupt dalla linea RST specificata.

Dati ad Ingresso Seriale – Questo bit rispecchia lo stato della linea SID.

RLC – RUOTA L'ACCUMULATORE A SINISTRA

RLC
07

Ruota i contenuti dell'Accumulatore a sinistra di un bit.
Si supponga che l'Accumulatore contenga $7A_{16}$ e lo stato Carry sia ad 1. Dopo che è stata eseguita l'istruzione:

RCL

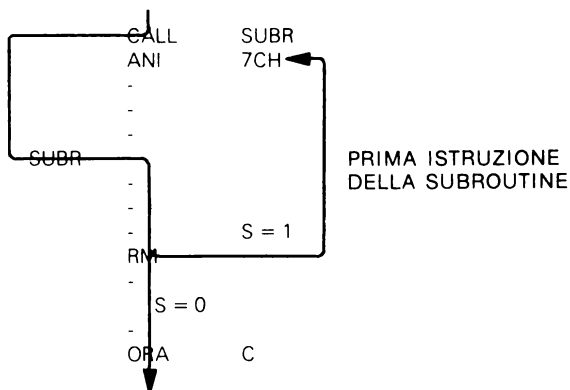
l'Accumulatore conterrà $F4_{16}$ e lo stato Carry sarà al livello logico 0:

Accumulatore	C		Accumulatore	C
01111010	1	→	11110100	0

RM — RITORNA SE LO STATO SIGN È UGUALE AD 1

RM
F8

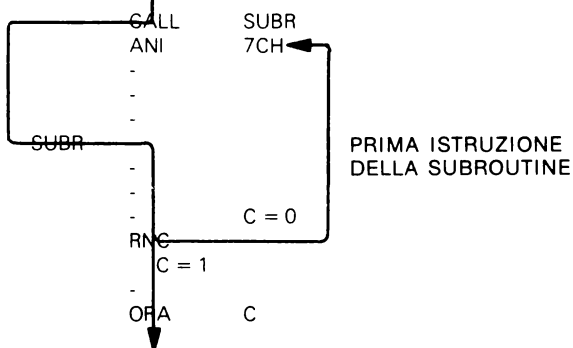
Questa istruzione è identica alla RET tranne il fatto che il ritorno non viene eseguito se lo stato Sign è 0, quando viene eseguita l'istruzione RM.

Si consideri la sequenza di istruzioni:

Dopo che è stata eseguita l'istruzione RM, se lo stato è uguale a 1, l'esecuzione ritorna all'istruzione ANI, che segue la CALL. Se lo stato Sign è uguale a 0 viene eseguita l'istruzione ORA che è la successiva in ordine sequenziale.

RNC — RITORNA SE LO STATO CARRY È UGUALE A 0

RNC
D0

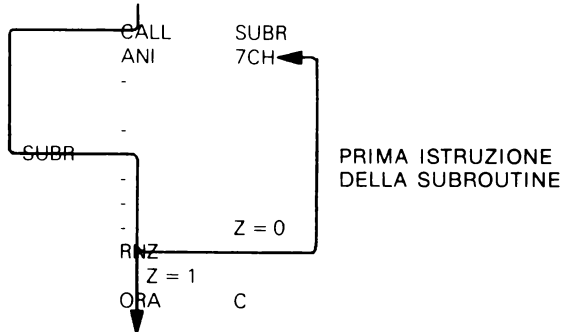
Questa istruzione è identica alla RET tranne il fatto che il ritorno non viene eseguito se lo stato Carry è uguale ad 1 quando viene eseguita l'istruzione RNC.

Si consideri la sequenza di istruzione:

Dopo che è stata eseguita l'istruzione RNC, se lo stato Carry è uguale a 0, l'esecuzione ritorna all'istruzione ANI, che segue la CALL. Se lo stato Carry è uguale ad 1 viene eseguita l'istruzione ORA, che è la successiva in ordine sequenziale.

RNZ — RITORNA SE LO STATO ZERO È UGUALE A 0

RNZ
CO

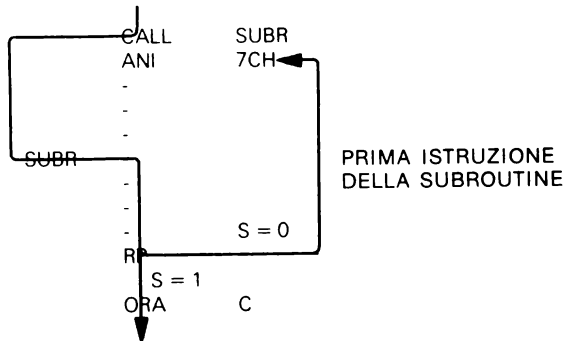
Questa istruzione è identica alla RET tranne il fatto che il ritorno non viene eseguito se lo stato Zero è uguale ad 1, quando viene eseguita l'istruzione RNZ.

Si consideri la sequenza di istruzioni:

Dopo che è stata eseguita l'istruzione RNZ, se lo stato Zero è uguale a 0, l'esecuzione ritorna all'istruzione ANI, che segue la CALL. Se lo stato Zero è uguale ad 1, viene eseguita l'istruzione ORA, che è la successiva in ordine sequenziale.

RP — RITORNA SE LO STATO SIGN È UGUALE A 0

RP
FO

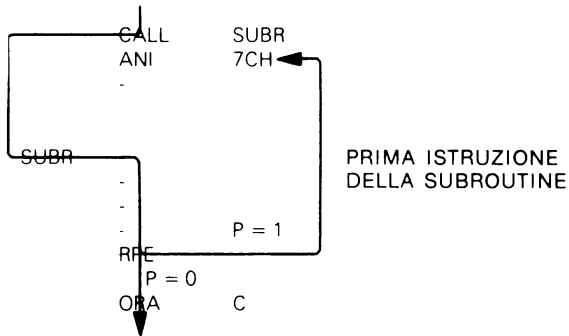
Questa è identica alla RET tranne il fatto che il ritorno non viene eseguito se lo stato Sign è uguale ad 1, durante l'esecuzione dell'istruzione RP.

Si consideri la sequenza di istruzioni:

Dopo che è stata eseguita l'istruzione RP, se lo stato Sign è uguale a 0, l'esecuzione ritorna all'istruzione ANI, che segue la CALL. Se lo stato Sign è uguale ad 1, viene eseguita l'istruzione ORA, che è la successiva in ordine sequenziale.

RPE — RITORNA SE LO STATO PARITÀ È UGUALE AD 1

RPE
E8

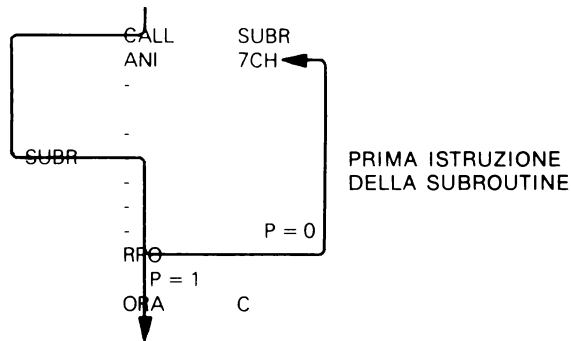
Questa istruzione è identica alla RET tranne il fatto che il ritorno non viene eseguito se lo stato Parità è uguale a 0, quando viene eseguita l'istruzione RPE.

Si consideri la sequenza di istruzioni:

Dopo che è stata eseguita l'istruzione RPE, se lo stato Parità è uguale ad 1, l'esecuzione ritorna all'istruzione ANI, che segue la CALL. Se lo stato Parità è uguale a 0, viene eseguita l'istruzione ORA, che è la successiva in ordine sequenziale.

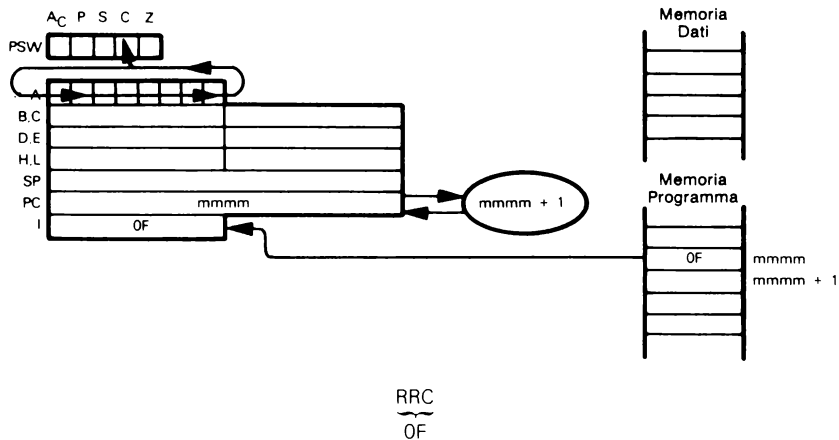
RPO — RITORNA SE LO STATO PARITÀ È UGUALE A 0

RPO
E0

Questa istruzione è identica alla RET tranne il fatto che il ritorno non viene eseguito se lo Stato Parità è uguale ad 1, durante l'esecuzione della RPE.

Si consideri la sequenza di istruzioni:

Dopo che è stata eseguita l'istruzione RPO, se lo stato parità è uguale a 0, l'esecuzione ritorna all'istruzione ANI, che segue la CALL. Se lo stato parità è uguale ad 1, viene eseguita l'istruzione ORA, che è la successiva in ordine sequenziale.

RRC — RUOTA L'ACCUMULATORE A DESTRA

RRC
⏟
0F

Ruota a destra di un bit i contenuti dell'Accumulatore.
Si supponga che l'Accumulatore contenga $7A_{16}$ e lo stato Carry sia ad 1. Dopo l'esecuzione dell'istruzione:

RRC

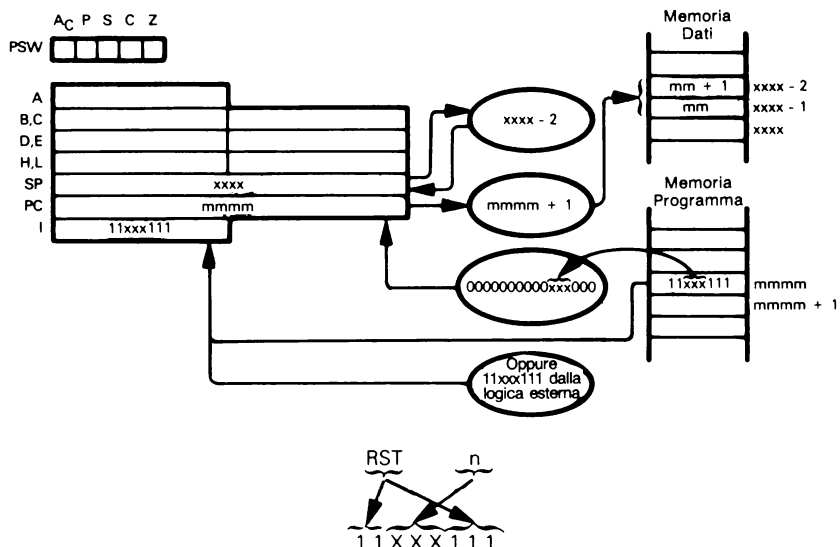
l'Accumulatore conterrà $3D_{16}$ e lo stato Carry sarà 0:

Accumulatore	C	→	Accumulatore	C
01111010	1		00111101	0

RRC potrebbe essere usata come istruzione logica.

RST — RESTART (RIPARTI)

RST n

1 1 X X X 1 1 1

Chiama la subroutine iniziante all'indirizzo di memoria specificato con n.

Quando viene eseguita l'istruzione:

RST 3

viene richiamata la subroutine iniziante alla locazione di memoria 0018_{16}. Il precedente contenuto del Contatore di Programma viene mandato alla sommità dello stack.

Normalmente l'istruzione RST viene impiegata assieme all'esecuzione di interrupt, come sarà descritto nel Capitolo 5.

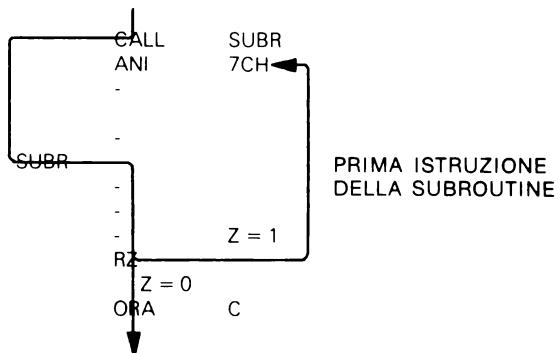
CHIAMATA DI SUBROUTINE USANDO RST

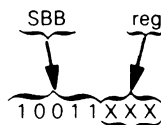
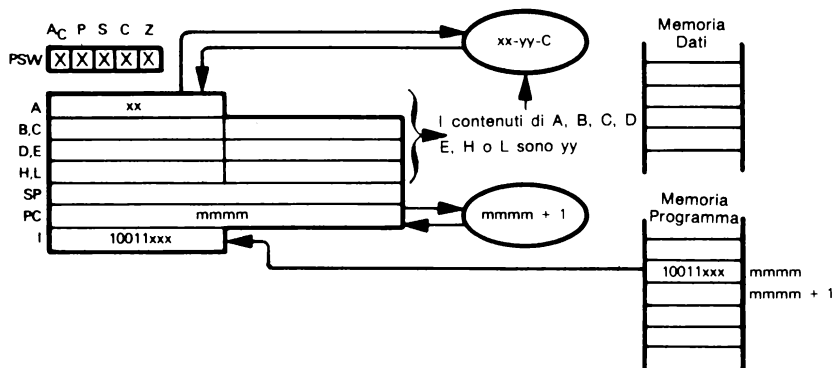
Se si hanno applicazioni che non fanno uso di tutti i codici istruzione RST per il servizio di interrupt non si apprezza la possibilità di chiamata di subroutine impiegando l'istruzione RST. Iniziando le subroutine ad appropriati indirizzi RST in fase di chiamata di queste subroutine è sufficiente un'istruzione a singolo byte RST al posto dell'istruzione CALL di tre byte.

RZ — RITORNA SE LO STATO ZERO È UGUALE AD 1

RZ
C8

Questa istruzione è identica alla RET tranne il fatto che il ritorno non viene eseguito se lo stato Zero è uguale a 0 quando si esegue l'istruzione RZ.

Si consideri la sequenza di istruzioni:

Dopo che è stata eseguita l'istruzione RZ, se lo stato Zero è uguale ad 1, l'esecuzione ritorna all'istruzione ANI, che segue la CALL. Se lo stato Zero è uguale a 0 viene eseguita l'istruzione ORA, che è la successiva in ordine sequenziale.

SBB — SOTTRAI UN REGISTRO O MEMORIA DALL'ACCUMULATORE CON PRESTITO

AC P S C Z
PSW X X X X X
A xx
B,C
D,E
H,L
SP
PC mmmm
I 10011xxx
xx-yy-C
I contenuti di A, B, C, D E, H o L sono yy
mmmm + 1
Memoria Dati
Memoria Programma
10011xxx mmmm
mmmm + 1

SBB reg

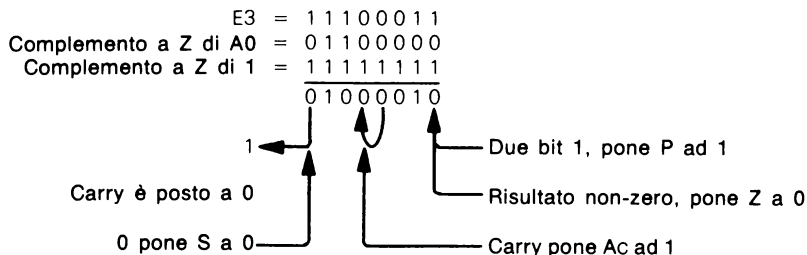
1 0 0 1 1 X X X

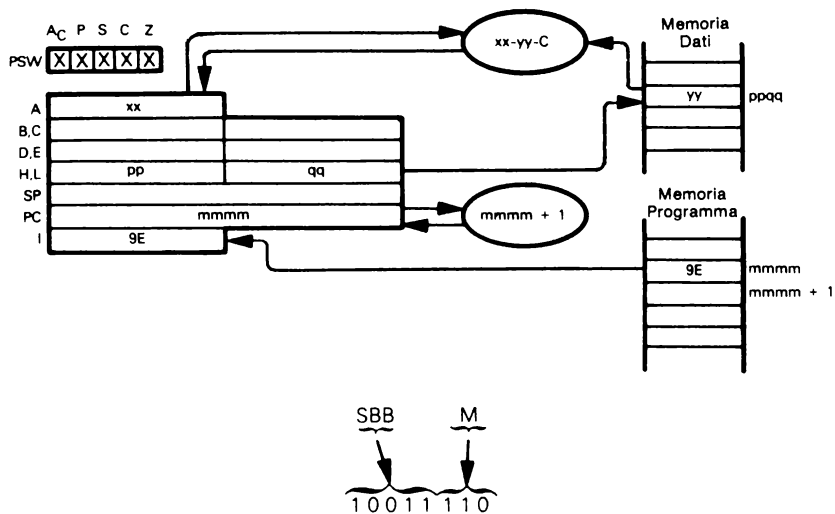
000 per reg = B
001 per reg = C
010 per reg = D
011 per reg = E
100 per reg = H
101 per reg = L
111 per reg = A

Sottrae i contenuti del registro specificato e lo stato Carry dall'Accumulatore, trattando i contenuti dei registri come semplici dati binari.

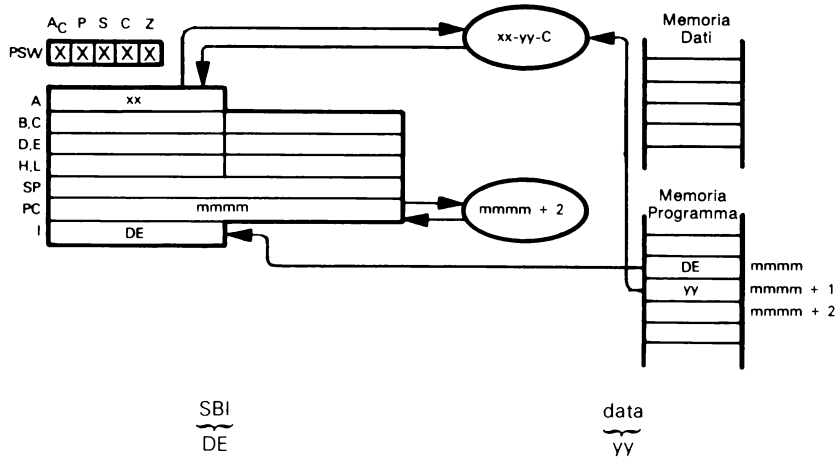
Si supponga xx = $E3_{16}$, E contenga $A0_{16}$, C = 1. Dopo l'esecuzione dell'istruzione:

SSB E

l'Accumulatore conterrà 42_{16}.

Anche i contenuti di un byte di memoria possono essere sottratti dall'Accumulatore:

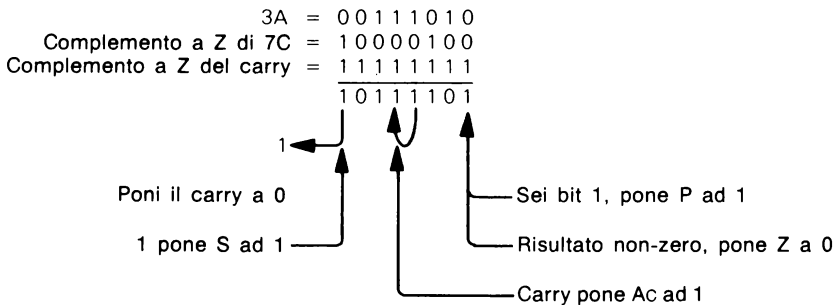
Se xx = $E3_{16}$, yy = $A0_{16}$ e C = 1 allora l'esecuzione dell'istruzione:

genera lo stesso risultato dell'esecuzione dell'istruzione SBB E, appena descritta.

L'istruzione SBB nella sottrazione multibyte dopo che i bit di basso ordine sono stati elaborati usando l'istruzione SUB.

SBI — SOTTRAI IMMEDIATO DATI DALL'ACCUMULATORE CON PRESTITO

Sottrae i contenuti del secondo byte di istruzione in codice e lo stato Carry dall'Accumulatore.

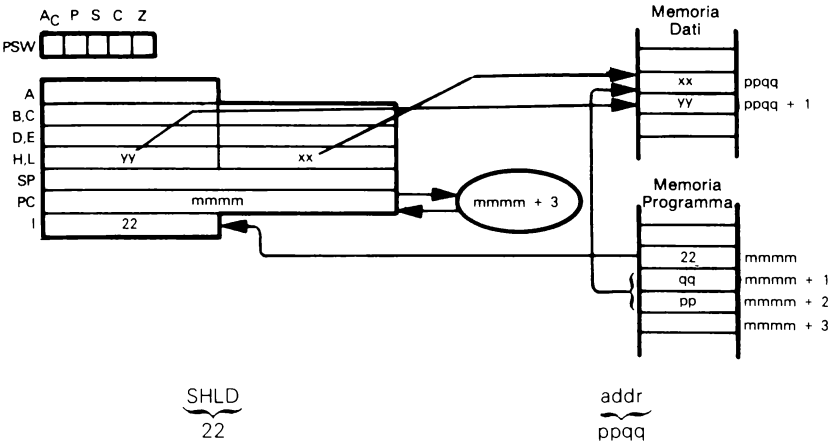
Si supponga xx = $3A_{16}$ e lo stato Carry sia uguale ad 1. Dopo che l'istruzione:

SBI 7CH

è stata eseguita, l'Accumulatore conterrà $8D_{16}$:

Si noti che il Carry risultante è complementato.

Questa istruzione non viene usata comunemente come SUI.

SHLD — MEMORIZZA DIRETTO I REGISTRI H ED L

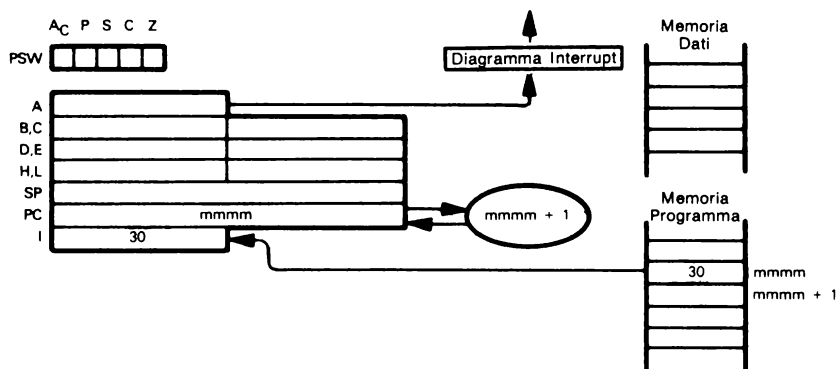
SHLD → 22   addr → ppqq

Il secondo e terzo byte in codice oggetto forniscono l'indirizzo di memoria del byte dati nel quale vengono scritti i contenuti del registro L. I contenuti del registro H vengono scritti nel byte dati sequenzialmente successivo.

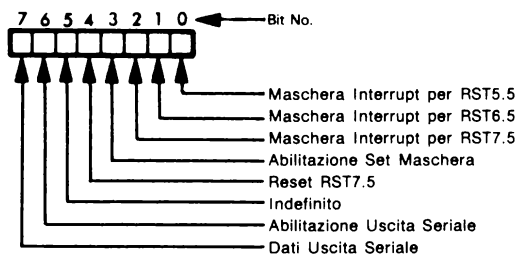
Si supponga xx = $2C_{16}$ ed yy = $3A_{16}$. Dopo che è stata eseguita l'istruzione:

```
LABEL  EQU    084AH
       —
       —
       —
       SHLD   LABEL
```

Il byte di memoria $084A_{16}$ conterrà $2C_{16}$ ed il byte di memoria $084B_{16}$ conterrà $3A_{16}$.

Si ricordi che EQU è un ordine assemblatore e non un'istruzione, esso dice all'Assemblatore di usare il valore a 16 bit $084A_{16}$ quando appare LABEL.

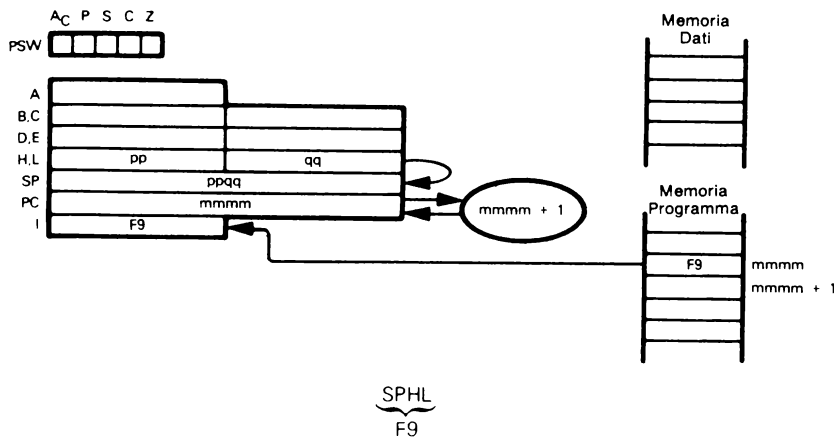
SIM – POSIZIONA MASCHERA INTERRUPT

SIM
30

Fa uscire i contenuti dell'Accumulatore per far ripartire la maschera di interrupt e la linea di uscita seriale (SOD). I contenuti dell'Accumulatore sono interpretati nel modo seguente:

La funzione maschera di interrupt viene eseguita solo se il bit di Abilitazione Set Maschera è 1. Per esempio se l'Accumulatore contiene OF_{16} e viene eseguita l'istruzione SIM, verrà inviato 1 al diagramma di interrupt per tutti i tre RST. Comunque se 07_{16}, fosse conservato nell'Accumulatore il diagramma di interrupt non sarebbe influenzato.

Se il diagramma di interrupt per un particolare RST è ad 1, tale RST è non abilitato, se a 0 è abilitato.

Il reset RST7.5 è usato, se il bit è ad 1, per azzerare un flip-flop di richiesta interna per RST7.5.

I dati di uscita seriale vengono eseguiti solo se il bit di Abilitazione Uscita Seriale è ad 1. Per esempio se l'Accumulatore conserva 03_{16} e viene eseguita l'istruzione SIM il SOD latch non verrà influenzato. Comunque se l'Accumulatore conserva 47_{16} e se viene eseguita l'istruzione SIM, verrà caricato uno 0 nel SOD latch.

SPHL — CARICA IL PUNTATORE DELLO STACK DAI REGISTRI H ED L

SPHL
F9

Muove i contenuti dei registri H ed L al Puntatore dello Stack. Si supponga pp = 08_{16} e qq = $3F_{16}$. Dopo che è stata eseguita l'istruzione:

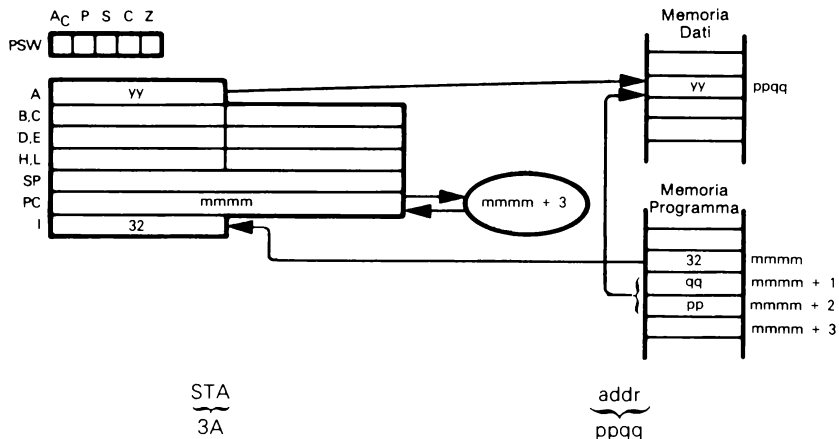
SPHL

il Puntatore dello Stack conterrà $083F_{16}$.

L'istruzione SPHL può essere usata per accedere a due stack — con un indirizzo conservato nei registri H ed L. Gli stacks sono spesso usati in questo modo per accedere alle stringhe di testo o dati qualsiasi ai cui bytes si deve accedere serialmente.

Il punto importante da ricordare è che questa logica dello stack può essere usata al posto dell'indirizzamento implicato di memoria con auto-incremento.

STA — IMMAGAZZINA L'ACCUMULATORE IN MEMORIA USANDO L'INDIRIZZAMENTO DIRETTO

STA — 3A addr — ppqq

Memorizza i contenuti dell'Accumulatore nel byte dell'istruzione STA in codice oggetto.

Si supponga che l'Accumulatore contenga $3A_{16}$. Dopo l'esecuzione dell'istruzione:

```
LABEL  EQU   084AH
       —
       —
       —
       STA   LABEL
```

il byte di memoria $084A_{16}$ conterrà $3A_{16}$.

Si ricordi che EQU è un ordine assemblatore e non una istruzione; esso dice all'Assemblatore di usare il valore a 16 bit $084A_{16}$ dovunque appare LABEL.

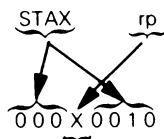
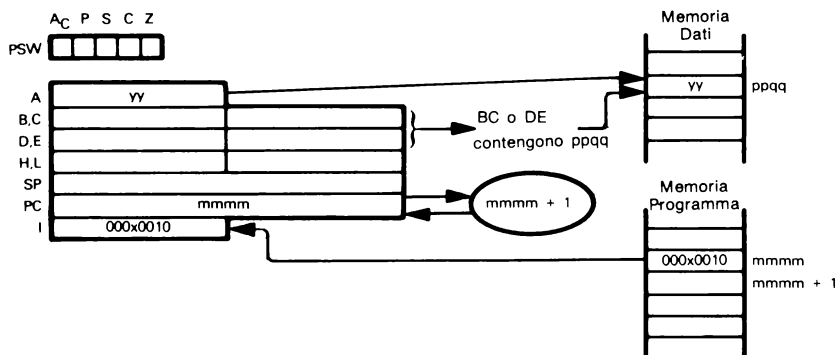
L'istruzione:

```
STA    LABEL
```

è equivalente alle due istruzioni:

```
LXI    H,LABEL
MOV    M,A
```

Quando si sta immagazzinando i valori di singoli dati in memoria è conveniente usare l'istruzione STA; essa usa un'istruzione e tre byte del programma oggetto per fare lo stesso che la combinazione LXI MOV fa in due istruzioni e quattro byte del programma oggetto. Inoltre la combinazione LXI MOV usa i registri H ed L, l'istruzione STA invece no.

STAX — IMMAGAZZINA IL CONTENUTO DELL'ACCUMULATORE NELLA LOCAZIONE DI MEMORIA INDIRIZZATA DA UNA COPPIA DI REGISTRI

0 se rp è B, rappresentante BC
1 se rp è D, rappresentante DE

Immagazzina i contenuti dell'Accumulatore nel byte di memoria indirizzato dalla coppia di registri BC o DE.

Si supponga che il registro B contenga 08_{16}, il registro C contenga $4A_{16}$ e l'Accumulatore contenga $3A_{16}$. Dopo che è stata eseguita l'istruzione:

```
STAX    B
```

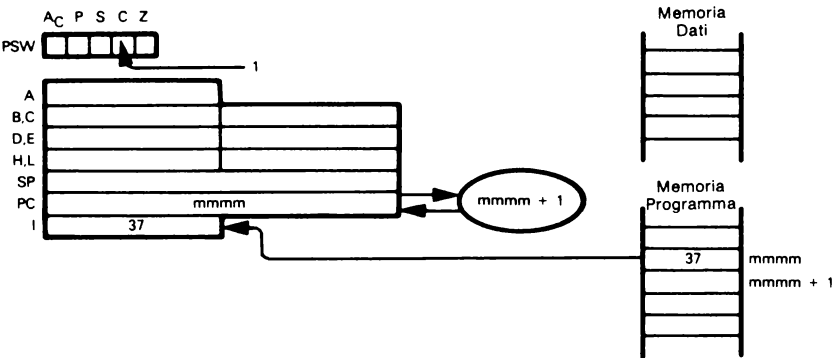
il byte di memoria $084A_{16}$ conterrà $3A_{16}$.

Si noti che non esiste l'istruzione STAX H perchè è identica all'istruzione MOV M,A.

Normalmente le istruzioni STAX ed LXI saranno usate assieme perché l'istruzione LXI carica un indirizzo a 16 bit nei registri BC o DE, come segue:

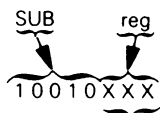
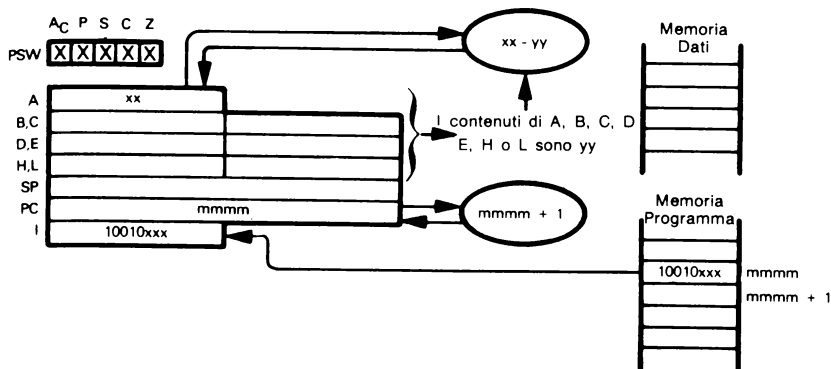
```
LXI     B,084AH
STAX    B
```

Si noti che l'istruzione STAX immagazzinerà dati solo dall'Accumulatore, mentre l'istruzione MOV immagazzinerà dati da qualunque registro.

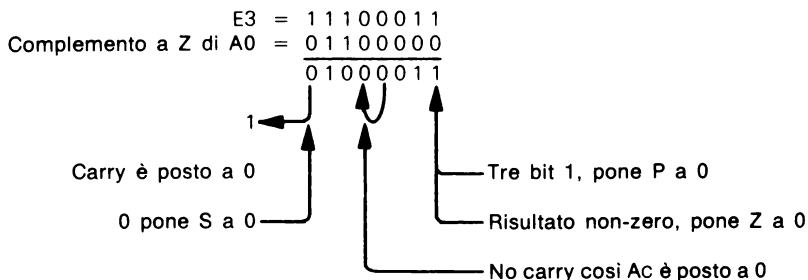
STC — PONI AD 1 LO STATO CARRY

$$\underbrace{\text{STC}}_{37}$$

Quando viene eseguita l'istruzione STC lo stato Carry è posto ad 1, indipendentemente dal suo precedente valore. Nessun altro stato o registro viene influenzato da questa istruzione.

SUB – SOTTRAI IL CONTENUTO DI UN REGISTRO O MEMORIA DALL'ACCUMULATORE

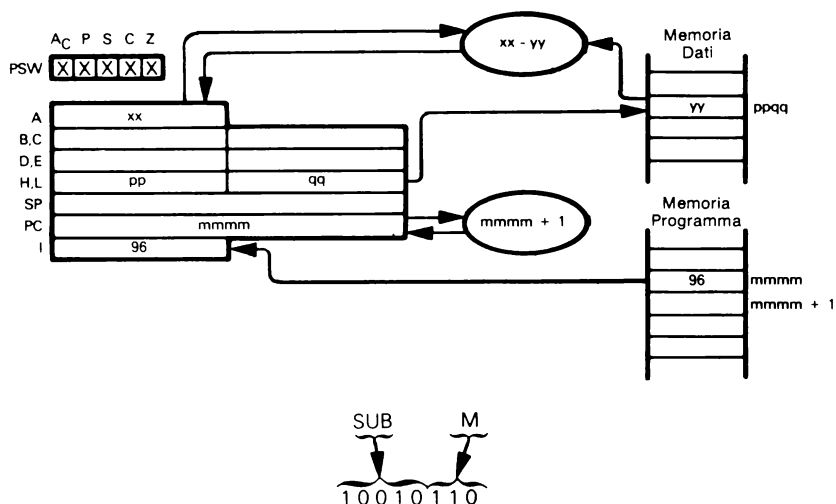
Questa istruzione assume due forme. La prima sottrae i contenuti di un registro dall'Accumulatore:

Sottrae i contenuti del registro specificato dall'Accumulatore, trattando i contenuti del registro come semplici dati binari. Si supponga xx = $E3_{16}$ ed il registro E contenga $A0_{16}$. Dopo l'esecuzione dell'istruzione:

SUB E

l'Accumulatore conterrà 43_{16}:

Si noti che il Carry risultante è complementato.

Anche i contenuti dei byte di memoria possono essere sottratti dall'Accumulatore:

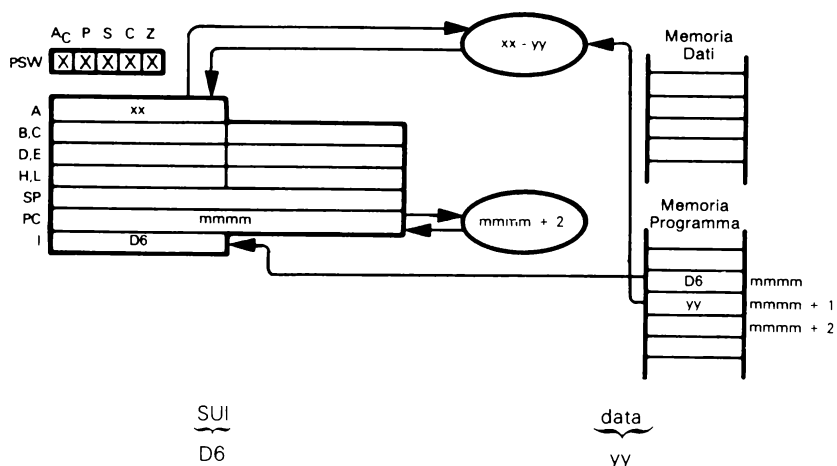
SUB M

10010110

Se xx = $E3_{16}$ ed yy = $A0_{16}$ allora l'esecuzione dell'istruzione:

SUB M

genera lo stesso risultato dell'esecuzione dell'istruzione SUB E, appena descritta.

L'istruzione SUB è usata per eseguire sottrazioni a byte singolo, o per i byte di basso ordine di sottrazioni multibyte.

SUI — SOTTRAI IMMEDIATO DATI DALL'ACCUMULATORE

Sottrae i contenuti del secondo byte del codice di istruzione dall'Accumulatore. Si supponga xx = $3A_{16}$. Dopo l'esecuzione dell'istruzione:

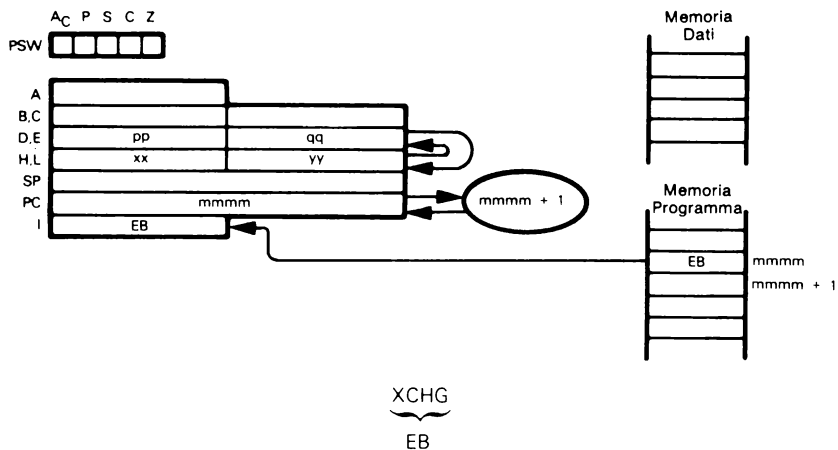
SBI 7CH

è stata eseguita, l'Accumulatore conterrà BE_{16}:

Si noti che il Carry risultante è complementato.

Questa istruzione è l'ideale per la sottrazione immediata.

XCHG — SCAMBIA I CONTENUTI DEI REGISTRI DE ED HL

$$\underbrace{\text{XCHG}}_{\text{EB}}$$

I contenuti dei registri D ed E sono scambiati con quelli dei registri H ed L. Si supponga pp = 03_{16}, qq = $2A_{16}$, xx = 41_{16} ed yy = FC_{16}. Dopo l'esecuzione dell'istruzione:

XCHG

H conterrà 03_{16}, L conterrà $2A_{16}$, D conterrà 41_{16} ed E conterrà FC_{16}.

Le due istruzioni:

```
XCHG
MOV    A,M
```

sono equivalenti a:

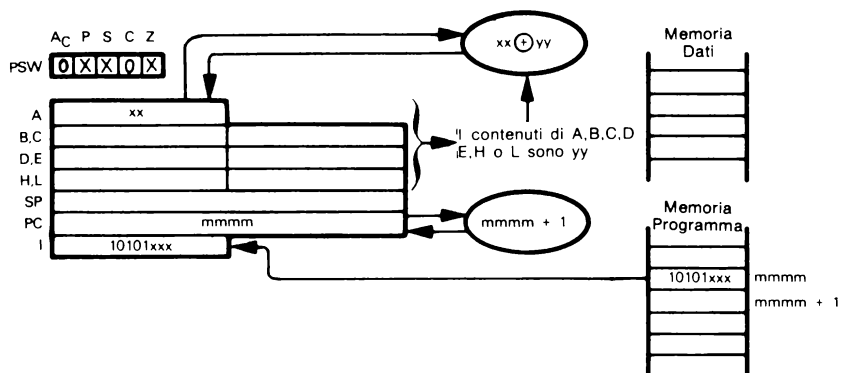
```
LDAX   D
```

Però se si vuole caricare l'indirizzo dati mediante i registri D ed E nel registro B, le due istruzioni:

```
XCHG
MOV    B,M
```

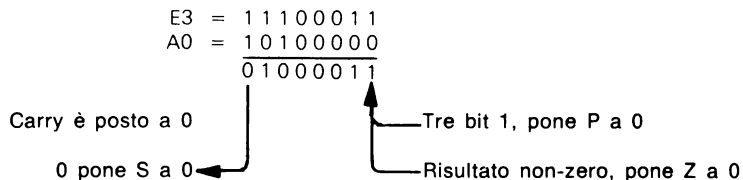
non hanno la singola istruzione equivalente.

XRA — OR ESCLUSIVO DI UN REGISTRO O MEMORIA CON L'ACCUMULATORE

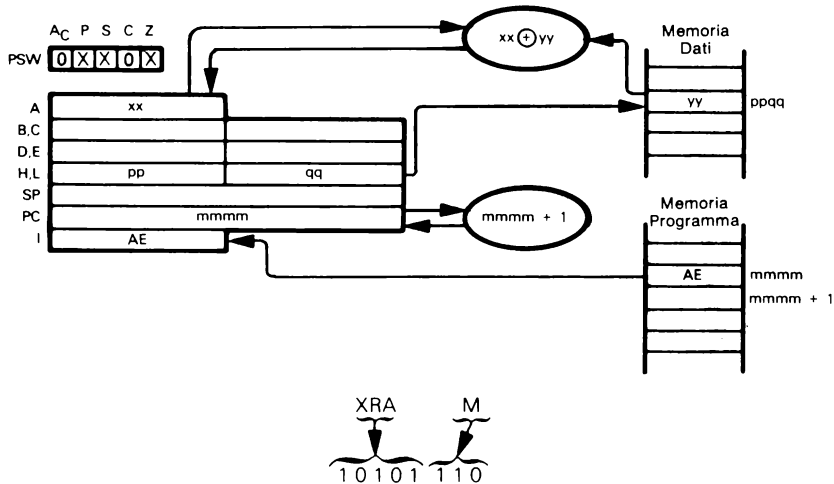
Questa istruzione assume due forme. La prima opera l'OR esclusivo dei contenuti di un registro con l'Accumulatore:

Opera l'OR esclusivo dei contenuti del registro specificato con l'Accumulatore, trattando i contenuti dei registri come semplici dati binari.

Si supponga xx = $3E_{16}$ ed il registro E contenga $A0_{16}$. Dopo l'esecuzione dell'istruzione:

XRA E

l'Accumulatore conterrà 43_{16}:

Si può operare l'OR esclusivo anche tra i contenuti di un byte di memoria e l'Accumulatore:

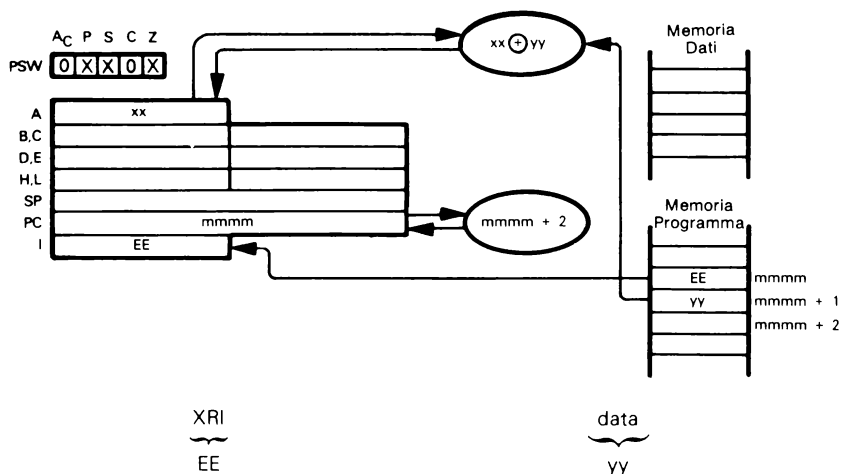
XRA M

10101 110

Se xx = $E3_{16}$ ed yy = $A0_{16}$ allora l'esecuzione dell'istruzione:

XRA M

genera lo stesso risultato dell'esecuzione dell'istruzione XRA E, appena descritta.

L'istruzione di OR esclusivo è usata come test per cambiare i bit di stato.

XRI — OR ESCLUSIVO IMMEDIATO DI DATI CON L'ACCUMULATORE

Opera l'OR esclusivo dei contenuti del secondo byte del codice di istruzione con l'Accumulatore. Si supponga xx = $3A_{16}$. Dopo l'esecuzione dell'istruzione:

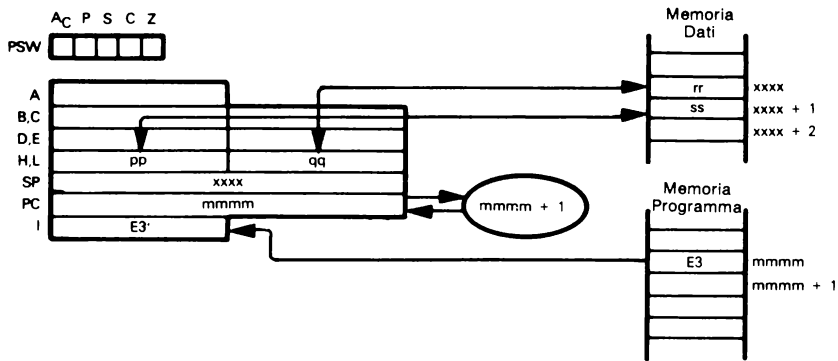
XRI 7CH

l'Accumulatore conterrà 46_{16}:

Questa è una istruzione logica di routine; viene impiegata spesso per la complementazione di bit. Per esempio l'istruzione

XRI 03H

complementerà incondizionatamente i due bit di basso ordine dell'Accumulatore.

XTHL – SCAMBIA LA SOMMITÀ DELLO STACK CON HL

XTHL
E3

Scambia i contenuti del registro H col byte alla sommità dello stack e quelli del registro H col byte sotto la sommità dello stack.

Si supponga pp = 21_{16}, qq = FA_{16}, rr = $3A_{16}$, ss = $E2_{16}$. Dopo l'esecuzione dell'istruzione:

XTHL

H conterrà $E2_{16}$, L conterrà $3A_{16}$ ed i due byte alla sommità dello stack conterranno rispettivamente FA_{16} e 21_{16}.

Le due istruzioni:

XTHL
XTHL

eseguite in sequenza si neutralizzano.

L'istruzione XTHL è usata per accedere e manipolare dati alla sommità dello stack come illustrato nella discussione sulle subroutine multiple del Capitolo 5.

CONVENZIONI DELL'ASSEMBLATORE INTEL 8080A ED 8085

L'assemblatore standard 8080A è disponibile dal costruttore dell'8080 e nelle più importanti reti time-sharing; esso è anche parte della maggioranza dei sistemi di sviluppo. Gli Assemblatori nella versione cross sono disponibili per i computer più grossi e molti minicomputer.

STRUTTURA DEL CAMPO ASSEMBLATORE

Le istruzioni in linguaggio assembly hanno la struttura di campo standard (vedere figura 2-1, Capitolo 2). I limiti richiesti sono:

1) Due punti dopo una label, tranne che per le pseudo-operazioni EQU, SET e MACRO che richiedono uno spazio.
2) Uno spazio dopo il codice operazione.
3) Una virgola tra operandi nel campo dell'operando. (Si ricordi questa richiesta!).
4) Un punto e virgola prima di un commento.

Istruzioni tipiche del linguaggio assembly 8080 sono:

```
START: LDA    1000   ;ACCETTA LUNGHEZZA
       LXI    H,2300
       HLT
```

LABEL

L'Assemblatore consente label di cinque caratteri; il primo carattere deve essere una lettera, @ oppure? — Gli altri caratteri possono essere lettere, numeri, @ oppure? — Qui si useranno solo lettere e numeri.

PSEUDO-OPERAZIONI

L'Assemblatore è dotato delle seguenti pseudo-operazioni:

```
DB    —    DEFINISCE BYTE
DS    —    DEFINISCE IMMAGAZZINAMENTO
DW    —    DEFINISCE PAROLA
END
EQU   —    EGUAGLIA
ORG   —    ORIGINE
SET
```

DB e DW sono le pseudo-operazioni DATI impiegate per posizionare i dati nelle ROM-DB è usato per dati ad 8 bit, DW per dati a 16 bit. L'unica caratteristica insolita da ricordare è che DW immagazzina gli otto bit meno significativi dei dati nella prima parola e gli otto bit più significativi nella seconda. Questa è una procedura standard, per memorizzare indirizzi in memoria, per l'8080A/8085, ma è contraria alla pratica comune; occorre quindi prestare molta attenzione all'ordine quando si memorizzano dati a 16 bit.

Esempi:

IND DW 3165H

risulta in (IND) = 65, e (IND + 1) = 31 (esadecimale).

TCOWV DB 32

Questa pseudo-operazione posiziona il numero 32 nel successivo byte di memoria ed assegna ad esso il nome TCONV.

ERROR: DB 'ERROR'

Questa pseudo-operazione posiziona i caratteri ASCII E, R, R, O ed R nei successivi 5 byte di memoria ed assegna il nome ERROR all'indirizzo del primo byte.

OPERS: DW FADD, FSUB, FMUL, FDIV

Questa pseudo-operazione posiziona gli indirizzi FADD, FSUB, FMUL, FDIV nei successivi otto bytes di memoria ed assegna il nome OPERS all'indirizzo del primo byte.

DS è la pseudo-operazione RESERVE, impiegata per l'assegnamento di locazioni in RAM; essa posiziona parole ad 8 bit (o byte).

EQU rappresenta la pseudo-operazione EGUAGLIA o DEFINISCI che viene usata per la definizione di nomi.

PSEUDO-OPERAZIONE EQU

SET è simile ad EQU, tranne il fatto che SET permette al nome di essere in seguito ridefinito. SET è poco comune e crea confusione; essa dovrebbe essere evitata a meno che non sia richiesta per unire programmi assieme.

PSEUDO-OPERAZIONE ORG

ORG è la pseudo-operazione standard INIZIA — ORG 0 posto all'inizio di un programma può essere omesso.

I programmi 8080A/8085 normalmente hanno diversi punti di inizio: essi sono impiegati come segue:

1) Per specificare l'indirizzo di RESET (normalmente zero).
2) Per specificare i punti di ingresso interrupt (normalmente da 0 a $3C_{16}$).
3) Per specificare l'indirizzo di inizio del programma principale.
4) Per specificare l'indirizzo di inizio delle subroutine.
5) Per definire l'immagazzinamento RAM.
6) Per definire uno Stack RAM

Esempio:

```
RESET  EQU   0
       ORG   RESET
```

Questa sequenza posiziona la sequenza di istruzione RESET in memoria cominciando dall'indirizzo 0.

```
INT 1  EQU   38H
       ORG   INT1
```

La sequenza di istruzione che segue è memorizzata in memoria a partire dalla locazione 38_{16}.

EN definisce semplicemente la fine del programma in linguaggio assembly.

LABEL CON PSEUDO-OPERAZIONI

Le regole e raccomandazioni per le label con pseudo-operazioni per l'8080A/8085 sono le seguenti:

1) EQU, SET e MACRO richiedono label poichè la funzione di queste pseudo-operazioni è quella di definire il significato di una label.
2) DB, DW e DS normalmente hanno le label (si noti che queste label non sono seguite dai due punti).
3) ORG, IF, ENDIF, ENDM ed END non dovrebbero avere label perchè il significato della label non è chiaro.

INDIRIZZI

NUMERI E CARATTERI NEL CAMPO INDIRIZZO

Gli assemblatori Intel 8080A/8085 permettono l'ingresso nel campo indirizzo a qualsiasi delle seguenti forme:

1) Decimale (una D finale è opzionale).
   Esempio: 1247 oppure 1247D.
2) Esadecimale (deve iniziare con un digit e terminare con una H).
   Esempio 124CH, 0E7H.
3) Ottale (deve terminare con O e Q ma Q genera meno confusione).
   Esempio: 1247Q oppure 1247O.
4) Binario (deve terminare con B).
5) ASCII (compreso entro virgolette).
   Esempio: 'HERE'.
6) Come un offset dal Contatore di Programma ($).
   Esempio: $+237H.

OPERAZIONI ARITMETICHE E LOGICHE DELL'ASSEMBLATORE

Tutte le operazioni aritmetiche e logiche dentro il campo indirizzo presuppongono che tutti gli argomenti siano dati a 16 bit; esse producono risultati a 16 bit. Queste operazioni sono consentite come parte di espressioni nel campo indirizzo:

1)	+	(addizione 16 bit)
2)	–	(sottrazione 16 bit)
3)	*	(moltiplicazione, generante un risultato a 16 bit)
4)	/	(quoziente intero)
5)	MOD	(parte intera)
6)	NOT	Operatori Booleani Standard
7)	AND	
8)	OR	
9)	XOR	
10)	SHR	(spostamento logico a destra)
11)	SHL	(spostamento logico a sinistra)

Gli spostamenti hanno la forma:

```
SHR     OP1,OP2
SHL     OP1,OP2
```

dove OP1 è il numero da spostare ed OP2 è il numero di spostamenti.

ORDINE DELLE OPERAZIONI DELL'ASSEMBLATORE

Per la risoluzione di espressioni con più di un operatore, di quelli elencati precedentemente, viene di seguito definito l'ordine di esecuzione. Operatori aventi lo stesso livello di precedenza sono valutati da sinistra a destra; espressioni in parentesi sono valutate prima. L'ordine di precedenza è:

Più alto: 1) *,/, MOD, SHL, SHR
2) +,−
3) NOT
4) AND
Più basso: 5) OR, XOR

Si raccomanda di evitare espressioni dentro il campo indirizzo, dovunque possibile. Se si deve calcolare un indirizzo si usino i seguenti accorgimenti:

1) Si usino le parentesi per rendere le espressioni più chiare. Non si faccia assegnamento sull'ordine delle operazioni.
2) Si commenti qualunque espressione non chiara.
3) Ci si assicuri che la valutazione dell'espressione non generi mai un risultato più largo di 16 bit.

ASSEMBLY CONDIZIONALE

Gli assemblatori 8080A/8085 hanno una semplice possibilità di assembly condizionale basato sulle pseudo-operazioni IF ed ENDIF — IF è seguito da una espressione, per esempio:

BASE-1000H oppure OPER1

Se l'espressione non è zero l'assemblatore comprende nel programma tutte le istruzioni fino ad ENDIF; se l'espressione è Zero l'assemblatore ignora tutte le istruzioni tra IF ed ENDIF. Qui non si useranno assemblaggi condizionati e non si farà riferimento a questa possibilità; essa è comoda talvolta per aggiungere od eliminare istruzioni di debugging del programma ma niente di più.

MACRO

Gli assemblatori standard 8080A/8085 hanno la possibilità di usare le macro per assegnare nomi alle sequenze di istruzione. Si usa la pseudo-operazione MACRO per iniziare la definizione ed ENDM per terminarla. La macro può avere dei parametri ed includere qualunque istruzione del linguaggio assembly eccetto la definizione di altre macro.

La possibilità macro è spesso una conveniente stenografia della programmazione, ma qui non verrà usata.

Si noti che le sequenze di istruzione definite mediante macro sono generalmente abbastanza corte; non si dovrebbero superare le dieci o quindici istruzioni. Sequenze più lunghe dovrebbero essere realizzate con subroutine per conservare spazio di memoria.

Ogni pseudo-operazione MACRO deve avere una label; questa label è il nome col quale si identifica la macro. Per una discussione su questo argomento si veda il Capitolo 2.

FORMATO BNPF

L'assemblatore 8080A/8085 produce nastri di carta perforata in un formato noto come BNPF. I dati binari sono rappresentati da un codice ASCII per le lettere B, N, P ed F come segue:

B	–	inizio della parola ad 8 bit
N	–	0 binario
P	–	1 binario
F	–	fine della parola ad 8 bit

Per esempio il codice binario 01101101 diventa:

BNPPNPPNPF

Così dieci byte del nastro di carta vengono impiegati per codificare un singolo byte dati. Questo formato viene impiegato per originare le maschere delle ROM e per programmare le PROM.

Capitolo 4
SEMPLICI PROGRAMMI

L'unico modo per apprendere la programmazione in linguaggio assembly è di scrivere programmi. Questo è quello che si farà nei prossimi sei capitoli, che contengono esempi di compiti del microprocessore. Dei problemi alla fine di ogni capitolo contengono delle variazioni rispetto agli esempi forniti nel corso del capitolo. Si dovrebbe provare l'esecuzione di esempi su microcomputer basati sull'8080A od 8085 per assicurarsi la comprensione del materiale contenuto nel capitolo.

In questo capitolo si inizia con alcuni programmi molto semplici.

FORMATO GENERALE DEGLI ESEMPI

FORMATO DELL'ESEMPIO

Ogni esempio di programma contiene le parti seguenti:

1) Un titolo che descrive il problema.
2) Uno statement di scopo che descrive il compito specifico che il pro-
3) Un semplice problema mostrante i dati di ingresso ed i risultati.
4) Un diagramma di flusso se la logica del programma è complessa.
5) Il programma sorgente, ovvero la lista del programma in linguaggio assembly.
6) Il programma oggetto, ovvero la lista del programma in linguaggio esadecimale di macchina.
7) Note di chiarimento che discutono le istruzioni ed i metodi impiegati nel programma.

I problemi alla fine del capitolo sono simili a quelli degli esempi; questi problemi dovrebbero essere programmati su un sistema a microcomputer basato sull'8080A od 8085, sulla falsariga degli esempi.

I programmi sorgente degli esempi sono stati costruiti come segue:

GUIDA AGLI ESEMPI

1) Vengono impiegate le notazioni standard degli assemblatori Intel 8085 ed 8080A, queste notazioni sono elencate al Capitolo 3.
2) Dati e codici indirizzo vengono selezionati per chiarezza piuttosto che consistenza. Si usa il sistema esadecimale per indirizzi di memoria, codici istruzione, e dati BCD, il sistema decimale per costanti numeriche, quello binario per diagrammi logici ed il sistema ASCII per i caratteri.
3) Spesso vengono descritte con enfasi istruzioni impiegate e tecniche di programmazione.
4) Gli esempi illustrano compiti che attualmente i microprocessori eseguono nei campi delle comunicazioni, strumentazione, computer, forniture commerciali, apparecchiature industriali e militari.
5) Vengono aggiunti commenti dettagliati.
6) Vengono descritte con enfasi le strutture semplici ma i programmi sono i più efficienti possibile nei limiti di questa guida. Le note spesso descrivono procedure più efficienti.
7) I programmi impiegano locazioni di memoria coerenti. Ogni programma inizia dalla locazione di memoria 0 (la locazione RESET) e termina con l'istruzione a ciclo senza fine:

```
HERE:   IMP   HERE
```

Questa istruzione a ciclo senza fine evita difficoltà associate con l'istruzione HLT dell'8080A (cioè sistemi senza interrupt è molto difficile ripristinare HLT). Si può sostituire questa istruzione con una istruzione di JUMP o RESTART che, in alcuni microcomputer basati sull'8080, trasferiscono il controllo sotto monitor. Si consulti il manuale dell'utente per il microcomputer in dotazione per la determinazione delle locazioni di memoria richiesta e l'istruzione di termine.

LINEE GUIDA PER I PROBLEMI

Quando si affrontano i problemi riportati alla fine di ogni capitolo si cerchi di lavorare all'interno delle seguenti linee guida:

LINEE GUIDA ALLA PROGRAMMAZIONE

1) Si commenti ogni programma cosicchè sia comprensibile ad altri.
   I commenti possono essere brevi e senza regole grammaticali; essi dovrebbero spiegare lo scopo di un'istruzione o di un gruppo del programma. I commenti non dovrebbero descrivere le operazioni od istruzioni che d'altra parte è disponibile sui manuali. Non è necessario commentare ogni statement o descrivere cose ovvie. Si può seguire il formato degli esempi ma fornire meno dettagli.
2) Si dia enfasi alla chiarezza, semplicità e buona struttura dei programmi. Se i programmi sono ragionevolmente efficienti non ci si preoccupi di economizzare una singola parola della memoria di programma o di pochi microsecondi.
3) Si costruiscano programmi ragionevolmente generali. Non si confondano parametri (come i numeri od elementi di un array) con costanti fisse (come π o ASCII C).
4) Non si assuma mai dei valori iniziali fissi per i parametri cioè si usi un'istruzione per caricare il valore iniziale di un parametro. Non si posizioni il valore iniziale di un parametro come parte del programma oggetto.
5) Si usino le notazioni dell'Assemblatore come mostrato negli esempi, notazioni che sono definite al Capitolo 3.
6) Si usino le notazioni esadecimali per gli indirizzi. Per i dati si usi la forma più chiara possibile.
7) Se il microcomputer in dotazione lo permette si inizino tutti i programmi dalla locazione di memoria 0 e si inizi dalla locazione di memoria 40_{16} per dati e memorizzazione temporanea. Altrimenti si stabiliscano indirizzi equivalenti per il microcomputer che si sta usando e lo si usi coerentemente. Inoltre si consulti il manuale dell'utente.
8) Si usino nomi significativi per le label e le variabili, es. SUM o CHECK piuttosto che X, Y o Z.
9) Si esegua ogni programma sul microcomputer. Non esiste altro modo per assicurare che il programma è corretto. Sono stati forniti dati campione per ogni problema. Ci si assicuri che il programma funzioni nei casi particolari.

Vengono di seguito elencate alcune informazioni da tenere ben presenti nella scrittura di programmi.

IMPIEGO DELL'ACCUMULATORE

La maggior parte delle istruzioni di processo (per esempio ADD, SUBTRACT AND, SHIFT, OR) impiegano l'Accumulatore. Nella maggior parte dei casi si caricheranno dati nell'Accumulatore sia con LDA che con MOV A, M. Si memorizzerà il risultato (dall'Accumulatore) in memoria con STA oppure con MOV M, A.

IMPIEGO DEL REGISTRO M

Il metodo migliore per l'accesso alla memoria è l'uso dell'indirizzamento implicato attraverso i registri H ed L, cioè l'uso del codice di registro M. Questo codice impone l'8080A ed 8085 all'esecuzione dell'Accesso in memoria impiegando l'indirizzo immagazzinato nei registri H ed L. Si può usare LXI (carica Immediato la coppia di registri) od LHLD (carica Diretto H ed L) per posizionare il valore iniziale nei Registri H ed L. Si può usare INX (Incrementa la Coppia di Registri) o DCX

(Decrementa la Coppia di Registri) per incrementare o decrementare (di 1) l'indirizzo nei registri H ed L.

Le operazioni aritmetiche e logiche ad 8 bit impiegano tutte l'Accumulatore come sede di uno degli operandi ed inoltre posizionano il risultato nell'Accumulatore.

ISTRUZIONI SPECIALI

Alcune operazioni aritmetiche e logiche hanno un impiego speciale, per esempio:

SUB A (oppure XRA A) azzera l'Accumulatore.

ADD A (sposta logicamente l'Accumulatore a sinistra). Questa istruzione inoltre moltiplica per 2 il contenuto dell'Accumulatore.

ANA A (oppure ORA A) azzera il flag Carry pur conservando il contenuto dell'Accumulatore.

Un AND logico può mascherare le parti di una parola. La maschera richiesta ha bit '1' nelle posizioni che si vuole conservare e bit '0' nelle posizioni che si vuole azzerare.

ESEMPI

Complemento ad uno

Scopo: Complementa in modo logico i contenuti della locazione di memoria 40_{16} e posiziona il risultato nella locazione di memoria 41_{16}.

Problema Campione:

(40) = 6A
Risultato = (41) = 95

Programma Sorgente:

```
       LDA    40H     ;PRENDE DATI
       CMA            ;COMPLEMENTA DATI
       STA    41H     ;IMMAGAZZINA IL RISULTATO
HERE:  JMP    HERE
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)	Contenuti di Memoria (Esadec.)
00	LDA 40H	3A
01		40
02		00
03	CMA	2F
04	STA 41H	32
05		41
06		00
07	HERE: JMP HERE	C3
08		07
09		00

LDA, STA e JMP richiedono tutte indirizzi a 16 bit; gli otto bit meno significativi dell'indirizzo si trovano nella parola immediatamente seguente il codice di istruzione e gli otto bit più significa-

tivi nella parola successiva (questo ordine è unico nell'8080 e contrario alla normale pratica dei calcolatori).

CMA è una parola di istruzione che inverte ogni bit dell'Accumulatore. Spesso si usa CMA per trasferire dati da oppure ad un dispositivo periferico impiegante logica negativa (per esempio un display che viene commutato on da uno zero, off da un uno).

L'indirizzo per l'istruzione «ciclo senza fine» (JMP HERE) è l'indirizzo della prima parola in questa istruzione. Sarà poi necessario ripristinare ovvero arrestare il microcomputer per farlo uscire dal ciclo.

Addizione ad 8 bit

Scopo: Somma i contenuti delle locazioni di memoria 40 e 41 e posiziona il risultato nella locazione di memoria 42.

Problema Campione:

```
      (40)  =  38
      (41)  =  2B
 Risultato  =  (42) 63
```

Programma Sorgente:

```
        LXI     H,40H
        MOV     A,M       ACCETTA IL PRIMO OPERANDO
        INX     H
        ADD     M         SOMMA IL SECONDO OPERANDO
        INX     H
        MOV     M,A       IMMAGAZZINA IL RISULTATO
HERE:   JMP     HERE
```

Programma oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)	Contenuti di Memoria (Esadec.)
00	LXI H,40H	21
01		40
02		00
03	MOV A,M	7E
04	INX H	23
05	ADD M	86
06	INX H	23
07	MOV M,A	77
08	HERE: JMP HERE	C3
09		08
0A		00

LXI H,40H carica i contenuti delle due parole seguenti della memoria di programma nella coppia di registri H (Registri H ed L). La prima parola va nel Registro L, la seconda in H.

Il codice registro 'M' significa che i dati sono ottenuti da, ovvero inviati alla locazione di memoria indirizzata dai registri H ed L. Così MOV A,M muove i dati dalla locazione indirizzata all'Accumulatore; MOV M,A muove i dati dall'Accumulatore alla locazione indirizzata; ADD M somma i contenuti della locazione indirizzata ai contenuti dell'Accumulatore. Si ricordi che H ed L contengono un indirizzo a 16 bit ma che la locazione di memoria contiene dati di 8 bit.

INX esegue un incremento a 16 bit di un ciclo di istruzione. La CPU non usa l'unità aritmetica ad 8 bit per l'incremento; essa usa l'incrementatore normalmente impiegato per incrementare il Contatore di Programma.

MOV A,M e MOV M,A sono preferibili alle istruzioni LDA e STA ogni volta che si usa ripetutamente la stessa locazione di memoria oppure si impieghino locazioni adiacenti. La convenienza sta nel fatto che MOV richiede meno memoria di programma e lo stesso tempo di LDA e STA. Si noti comunque che si devono caricare i registri H ed L prima di poter impiegare il registro M.

Spostamento a Sinistra di un Bit

Scopo: Eseguire lo spostamento di un bit dei contenuti della locazione di memoria 40 e posizionare il risultato nella locazione di memoria 41. Azzerare la posizione del bit non occupata.

Problema Campione:

(40) = 6F
Risultato = (41) = DE

Programma Sorgente:

```
        LDA     40H     ;ACCETTA I DATI
        ADD     A       ;SPOSTA I DATI A SINISTRA LOGICAMENTE
        STA     41H     ;IMMAGAZZINA IL RISULTATO
HERE:   JMP     HERE
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)	Contenuti di Memoria (Esadec.)
00	LDA 40H	3A
01		40
02		00
03	ADD A	87
04	STA 41H	32
05		41
06		00
07	HERE: JMP HERE	C3
08		07
09		00

ADD A Somma semplicemente il contenuto dell'Accumulatore a sè stesso. Il risultato è, naturalmente, il doppio dei dati originali, che è lo stesso risultato che produrrebbe uno spostamento logico a sinistra. Il bit di minor significato del risultato è zero poichè 0+0 = 1+1 = 0; 1+1 genera però anche un Carry al bit successivo.

Mascheratura dei Quattro Bit Più Significativi

Scopo: Posizionare i quattro bit meno significativi del contenuto della locazione di memoria 40 nella locazione di memoria 41. Inoltre cancella i quattro bit più significativi della locazione di memoria 41.

Problema Campione:

(40) = B8
Risultato = (40) = 08

Programma Sorgente:

```
       LDA    40H         ;ACCETTA DATI
       ANI    00001111B   ;MASCHERATURA DEI BIT PIÙ SIGNIFICATIVI
       STA    41H         ;IMMAGAZZINA IL RISULTATO
HERE:  JMP    HERE
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)	Contenuti di Memoria (Esadec.)
00	LDA 40H	3A
01		40
02		00
03	ANI 00001111B	E6
04		0F
05	STA 41H	32
06		41
07		00
08	HERE: JMP HERE	C3
09		08
0A		00

L'istruzione ANI opera l'AND logico dei contenuti dell'Accumulatore con i contenuti della parola della memoria di programma immediatamente seguente l'istruzione. L'AND può essere usato per cancellare bit che non servono. I quattro bit meno significativi potrebbero essere un ingresso da un commutatore ovvero una uscita ad un display numerico.

Cancellazione di una Locazione di Memoria

Scopo: Cancella la locazione di memoria 40

Programma Sorgente:

```
        SUB     A
        STA     40H     ;CANCELLA LA LOCAZIONE 40
HERE:   JMP     HERE
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)	Contenuti di Memoria (Esadec.)
00	SUB A	97
01	STA 40H	32
02		40
03		00
04	HERE: JMP HERE	C3
05		04
06		00

SUB A sottrae il numero contenuto nell'Accumulatore a sè stesso. Il risultato è l'azzeramento dell'Accumulatore. Questa istruzione, XRA A oppure MVI A,0 possono tutte azzerare l'Accumulatore. MVI A,0 impiega però più tempo e memoria ma non influenza i flag di stato.

Separazione di una Parola

Scopo: Divide i contenuti della locazione di memoria 40 in due parti di 4 bit e li immagazzina nelle locazioni di memoria 41 e 42. Posiziona i quattro bit più significativi della locazione 40 nei quattro bit della posizione meno significativa della locazione di memoria 41; inoltre posiziona i quattro bit meno significativi nella locazione di memoria 40 nella posizione meno significativa della locazione di memoria 42. Cancella i quattro bit più significativi delle locazioni di memoria 41 e 42.

Problema Campione:

```
     (40)  =  3F
Risultato  =  (41) = 03
              (42) = 0F
```

Programma Sorgente:

```
LXI     H,40H     ;ACCETTA I DATI
MOV     A,M
MOV     B,A
RRC
RRC
RRC               ;SPOSTA I DATI A DESTRA 4 VOLTE
RRC
ANI     00001111B;MASCHERA OFF I BIT PIÙ SIGNIFICATIVI
INX     H
MOV     M,A       ;IMMAGAZZINA I BIT PIÙ SIGNIFICATIVI
MOV     A,B       ;RIMEMORIZZA I DATI ORIGINARI
ANI     00001111B;MASCHERA OFF I BIT MENO SIGNIFICATIVI
INX     H
MOV     M,A       ;IMMAGAZZINA I BIT MENO SIGNIFICATIVI
HERE:
IMP
HERE
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)			Contenuti di Memoria (Esadec.)
00		LXI	H,40H	21
01				40
02				00
03		MOV	A,M	7E
04		MOV	B,A	47
05		RRC		0F
06		RRC		0F
07		RRC		0F
08		RRC		0F
09		ANI	00001111B	E6
0A				0F
0B		INX	H	23
0C		MOV	M,A	77
0D		MOV	A,B	78
0E		ANI	00001111B	E6
0F				0F
10		INX	H	23
11		MOV	M,A	77
12	HERE:	JMP	HERE	C3
13				12
14				00

Le istruzioni che impiegano il Registro M occupano solo una parola della memoria di programma. Comunque il Registro di Indirizzo (H od L) deve essere caricato prima che M possa essere usato. Così l'indirizzamento implicato di memoria permette un risparmio di tempo e memoria, rispetto all'indirizzamento diretto della memoria, solo se la ripetibilità del programma impiega gli stessi indirizzi ovvero indirizzi consecutivi.

L'istruzione RRC sposta l'Accumulatore a destra di un bit, circolarmente, andando dal bit della posizione meno significativa a quello della posizione più significativa del Carry. Lo spostamento dell'Accumulatore a destra di 4 bit richiede quattro istruzioni RRC.

Molte istruzioni dell'8080A/8085 influenzano una coppia di registri ad 8 bit. Le coppie sono H (H ed L) D (D ed E), B (B e C) ed SP (il Puntatore dello Stack). I registri B, D e H rappresentano gli 8 bit più significativi. Le istruzioni più comuni che usano una coppia di registri sono: LXI (Carica Immediato una Coppia di Registri), DCX (Decrementa a 16 bit), INX (Incrementa una Coppia di Registri) e DAD (Somma una coppia di Registri ad H ed L.)

Ricerca del Maggiore di Due Numeri

Scopo: Posiziona il maggiore dei contenuti delle locazioni di memoria 40 e 41 nella locazione di memoria 42. Si assuma che i contenuti delle locazioni di memoria 40 e 41 rappresentano tacitamente numeri binari.

Problemi Campione:

```
a.          (40)  =  3F
            (41)  =  2B
       Risultato  =  (42) = 3F
b.          (40)  =  75
            (41)  =  A8
       Risultato  =  (42) = A8
```

Programma Sorgente:

```
        LXI    H,40H
        MOV    A,M      ;ACCETTA IL PRIMO OPERANDO
        INX    H
        CMP    M        ;IL SECONDO OPERANDO È MAGGIORE?
        JNC    DONE
        MOV    A,M      ;SÌ, ALLORA PRENDI IL SECONDO OPERANDO
DONE:   INX    H
        MOV    M,A      ;MEMORIZZA L'OPERANDO MAGGIORE
HERE:   JMP    HERE
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)			Contenuti di Memoria (Esadec.)
00		LXI	H,40H	21
01				40
02				00
03		MOV	A,M	7E
04		INX	H	23
05		CMP	M	BE
06		JNC	DONE	D2
07				0A
08				00
09		MOV	A,M	7E
0A	DONE:	INX	H	23
0B		MOV	M,A	77
0C	HERE:	JMP	HERE	C3
0D				0C
0E				00

CMP M posiziona i flags come se i contenuti delle locazioni di memoria indirizzata da H ed L fossero sottratti ai contenuti dell'Accumulatore. Comunque i contenuti dell'Accumulatore sono mantenuti invariati per successivi confronti od altre elaborazioni.

Se A è il contenuto dell'Accumulatore ed X è il secondo operando per le istruzioni CM e CPI allora i flag sono posizionati come segue:

1) Zero = 1 se $A = X$
   Zero = 0 se $A \neq X$
2) Carry = 1 se $A < X$
   Carry = 0 se $A \geq X$
   (A,X sono considerati tacitamente numeri binari)

CMP pone il Carry ad 1 se un Prestito fosse necessario per eseguire attualmente la sottrazione, cioè se il numero da sottrarre ai contenuti dell'Accumulatore fosse maggiore di quei contenuti. Così la sequenza di CMP, JNC origina un salto se i contenuti dell'Accumulatore sono maggiori o uguali all'altro numero.

JNC DONE origina un salto alla locazione di memoria DONE se il flag Carry = 0. Diversamente (se Carry = 1) il calcolatore procede con la successiva locazione di memoria, in ordine sequenziale, dopo l'istruzione JNC.

DONE ed HERE sono nomi appropriati da assegnare alle locazioni della memoria di programma in quanto sono facili da ricordare. Si ricordi che questi nomi sono label e quindi devono essere eseguiti da due punti nella riga dove sono definiti. In questo programma DONE è la locazione di memoria 000A ed HERE la locazione 000C. Gli assemblatori 8080A/8085 consentono cinque caratteri per la label – il primo deve essere una lettera mentre gli altri possono essere lettere e numeri (sono permessi alcuni caratteri speciali ma qui non verranno usati.)

Addizione a 16 Bit

Scopo: Somma il numero a 16 bit delle locazioni di memoria 40 e 41 al numero a 16 bit delle locazioni di memoria 42 e 43. Gli 8 bit più significativi dei numeri da sommare si trovano nelle locazioni di memoria 41 e 43. Immagazzina il risultato nelle locazioni di memoria 44 e 45 con il byte più significativo nella locazione di memoria 45.

Problema Campione:

```
      (40)  =  2A
      (41)  =  67
      (42)  =  F8
      (43)  =  14
 Risultato  =  672A = 14F8 = 7C22
      (44)  =  22
      (45)  =  75
```

Programma Sorgente:

```
        LHLD    40H     ;ACCETTA IL PRIMO NUMERO A 16 BIT
        XCHG
        LHLD    42H     ;ACCETTA IL SECONDO NUMERO A 16 BIT
        DAD     D       ;ADDIZIONE A 16 BIT
        SHLD    44H     ;IMMAGAZZINA IL RISULTATO A 16 BIT
HERE:   JMP     HERE
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)	Contenuti di Memoria (Esadec.)
00	LHLD 40H	2A
01		40
02		00
03	XCHG	EB
04	LHLD 42H	2A
05		42
06		00
07	DAD D	19
08	SHLD 44H	22
09		44
0A		00
0B	HERE: JMP HERE	C3
0C		0B
0D		00

LHLD carica i Registri H ed L da due locazioni di memoria, una specificata nell'istruzione e l'altra è la successiva. I contenuti della prima locazione di memoria indirizzata vanno al Registro L quelli della successiva locazione al registro H. Così LHLD 40 significa (L) = (40), (H) = (41). Il trasferimento attuale procede 1 bit alla volta ed impiega 16 cicli di clock. Il vantaggio di un'istruzione di caricamento a 16 bit rispetto a due istruzioni di caricamento ad 8 bit sta nel fatto che la CPU compie una sola fase di esecuzione di istruzione dalla memoria. Si noti che la differenza tra LXI, che carica un valore fisso a 16 bit da ROM in una coppia di registri ed LHLD, che carica i contenuti di due locazioni RAM in H ed L.

XCHG scambia i contenuti dei registri D ed E, con H ed L. Nessun numero viene cambiato o ruotato a destra. XCHG può sostituire un'intera serie di istruzioni MOV.

DAD somma i numeri a 16 bit dei Registri D ed E con H ed L. Il risultato è posizionato nei registri H ed L. DAD attualmente somma un bit alla volta; essa impiega dieci cicli di clock.

SHLD immagazzina i contenuti dei registri H ed L in due locazioni di memoria, la prima specificata nell'istruzione e la seconda nella istruzione successiva. I contenuti di L vanno nella locazione specificata ed i contenuti di H vanno nella locazione successiva. Così SHLD 44 significa (44) = (L), (45) = (H). Analogamente alla LHLD il trasferimento attuale procede un bit alla volta; esso viene eseguito in 16 cicli di clock.

Sebbene l'8080 sia un microprocessore ad 8 bit esso è dotato di istruzioni che possono maneggiare numeri a 16 bit. Queste istruzioni sono soprattutto intese a maneggiare indirizzi ma possono essere impiegate per dati a 16 bit. Le più comuni ed i loro impieghi sono:

1) DAD — Addizione a 16 bit
   Impiegata per accedere a tabelle e per sommare dati a 16 bit.
2) DCX — Decremento a 16 bit
   Impiegata per sottrarre 1 ai contenuti di un registro Indirizzo.
3) INX — Incremento a 16 bit
   Impiegata per sommare 1 ai contenuti di un registro Indirizzo.
4) LHLD — Caricamento Diretto a 16 bit
   Impiegata per posizionare indirizzi variabili nel Registro del Corpo Indirizzo (H ed L).
5) LXI — Caricamento Immediato a 16 bit
   Impiegata per inizializzare un Registro Indirizzo con un valore fisso, cioè l'indirizzo di partenza di un array o di una tabella.
6) SHLD — Immagazzinamento Diretto a 16 bit
   Impiegata per memorizzare indirizzi in memoria da un registro del corpo Indirizzo (H ed L).

Tabella dei Quadrati

Scopo. Calcolare il quadrato dei contenuti della locazione di memoria 40, impiegando una tabella dei quadrati e posizionando il risultato nella locazione di memoria 41. Si assuma che la locazione di memoria 40 contenga un numero tra 0 e 7 compresi, cioè $0 \leq (40) \leq 7$.

La tabella occupi le locazioni di memoria da 60 a 67.

Indirizzo di Memoria (Esadec.)	Ingresso (Esadecimale)	(Decimale)	
60	00	0	(0^2)
61	01	1	(1^2)
62	04	4	(2^2)
63	09	9	(3^2)
64	10	16	(4^2)
65	19	25	(5^2)
66	24	36	(6^2)
67	31	49	(7^2)

Problemi Campione:

a. (40) = 03
Risultato = (41) = 09

b. (40) = 06
Risultato = (41) = 24

Programma Sorgente:

```
       LDA   40H     ;ACCETTA DATI
       MOV   L,A     ;CONTRASSEGNA I DATI IN UN INDICE A 16 BIT
       MVI   H,0
       LXI   D,60H   ;ACCETTA L'INDIRIZZO DI PARTENZA
                     ;DELLA TABELLA DEI QUADRATI
       DAD   D       ;INDICE TABELLA CON DATI
       MOV   M       ;ACCETTA IL QUADRATO DEI DATI
       STA   41,H
HERE:  JMP   HERE
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)			Contenuti di Memoria (Esadec.)
00		LDA	40H	3A
01				40
02				00
03		MOV	L,A	6F
04		MVI	H,0	26
05				00
06		LXI	D,60H	11
07				60
08				00
09		DAD	D	19
0A		MOV	A,M	7E
0B		STA	41H	32
0C				41
0D				00
0E	HERE:	JMP	HERE	C3
0F				0E
10				00

Si noti che si deve anche posizionare la tabella dei quadrati in memoria (la pseudo-operazione DB dell'assemblatore opererà questo). La tabella dei quadrati rappresenta dei dati costanti, non parametri che possono cambiare; per questo si può inizializzare la tabella dei quadrati impiegando la pseudo-operazione DB, piuttosto che l'esecuzione di istruzioni di caricamento dei valori in tabella.

MOV L,A muove i dati nell'Accumulatore dal Registro L. I dati sono gli otto bit meno significativi dell'indice.

MVI H,0 azzera il Registro H così che esso non interferisce con la somma a 16 bit dell'indirizzo di partenza ed esponente. Non si assuma mai che un registro contenga zero all'inizio di un programma.

LXI D,60H carica l'indirizzo di partenza della tabella nei registri D ed E. Si usano D ed E per l'indirizzo di partenza perchè l'istruzione DAD non varia questi registri. Così l'indirizzo di partenza della tabella sarà rimasto in D ed E nell'eventualità che si desideri un altro elemento dalla tabella.

DAD D somma l'indirizzo di partenza e l'indice; il risultato in H ed L è così corretto. MOV A,M poi muove questo ingresso all'Accumulatore.

TABELLE DI CONSULTAZIONE

Operazioni aritmetiche che un microprocessore non può eseguire direttamente in poche istruzioni sono spesso meglio eseguite con tabelle di consultazione. Queste tabelle contengono semplicemente tutte le possibili risposte al problema; esse sono organizzate in modo che la risposta ad un particolare problema può essere trovata con facilità. I problemi aritmetici così diventano un problema di accesso — come prelevare la risposta corretta dalla tabella? Occorre conoscere due cose — la posizione della risposta nella tabella (detto indice) e la base o indirizzo di partenza della tabella. L'indirizzo della risposta è poi proprio l'indirizzo della base più l'indice. L'indirizzo della base, naturalmente è un numero fisso per una data tabella. Come si può determinare invece l'indice? In casi semplici, dove un singolo elemento dati è coinvolto si può organizzare la tabella in modo che i dati siano l'indice. Nella tabella dei quadrati l'ingresso 0 corrisponde a 0 al quadrato, il primo ingresso 1 al quadrato, ecc. Nei casi più complessi dove la variazione degli ingressi è molto più grande oppure ci sono molti dati (per esempio radici di un quadrato o numero delle permutazioni) occorre determinare l'indice usando metodi più complicati.

Il compromesso nell'impiego di una tabella è tempo-memoria. Le tabelle sono più veloci non essendo richiesti calcoli, e più semplici poichè non è necessario escogitare e verificare metodi matematici. Comunque le tabelle possono impiegare una grande quantità di memoria se il range dei dati d'ingresso è ampio. Si può spesso ridurre le dimensioni della tabella limitando la precisione del risultato, diminuendo i dati d'ingresso ovvero organizzando ingegnosamente la tabella. Le tabelle vengono spesso impiegate nel calcolo di funzioni trascendenti o trigonometriche, linearizzare ingressi, convertire codici ed eseguire altri compiti di tipo matematico.

Complemento ad Uno a 16 bit

Scopo: Posiziona il complemento ad uno di un numero a 16 bit che risiede nelle locazioni di memoria 40 e 41, nelle locazioni di memoria 42 e 43. I byte più significativi si trovano nelle locazioni di memoria 41 e 43.

Problema Campione:

```
      (40)  =  67
      (41)  =  E2
 Risultato  =  (42) = 9 8
               (43) = 1 D
```

Il complemento ad uno inverte ogni bit del numero originale; la somma del complemento ad uno e del numero originale sarà sempre formata da bit tutti uguali ad 1.

Programma Sorgente:

```
       LHLD   40H    ;ACCETTA I DATI
       MOV    A,L    ;COMPLEMENTA GLI 8 BIT MENO SIGNIFICATIVI
       CMA
       MOV    L,A
       MOV    A,H    ;COMPLEMENTA GLI 8 BIT PIÙ SIGNIFICATIVI
       CMA
       MOV    H,A
       SHLD   42H    ;IMMAGAZZINA IL COMPLEMENTO AD UNO
HERE:  JMP    HERE
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)	Contenuti di Memoria (Esadec.)
00	LHLD 40H	2A
01		40
02		00
03	MOV A,L	7D
04	CMA	2F
05	MOV L,A	6F
06	MOV A,H	7C
07	CMA	2F
08	MOV H,A	67
09	SHLD 42H	22
0A		42
0B		00
0C	HERE: JMP HERE	C3
0D		0C
0E		00

PROBLEMI

1) Complemento a Due

Scopo: Posiziona il complemento a due dei contenuti della locazione di memoria 40 nella locazione di memoria 41. Il complemento a due è il complemento ad uno più uno.

Problema Campione:

(40) = 3E
Risultato = (41) = D2

La somma del numero originale ed il suo complemento a due è zero (si verifichi il caso campione).

2) Sottrazione ad 8 Bit

Scopo: Sottrae i contenuti della locazione di memoria 41 dai contenuti della locazione di memoria 40. Posiziona il risultato nella locazione di memoria 42.

Problema Campione:

(40) = 77
(41) = 39
Risultato = (42) = 3E

3) Spostamento a Sinistra di due Bit

Scopo: Sposta i contenuti della locazione di memoria 40 di due bit a sinistra e posiziona il risultato nella locazione di memoria 41. Azzera le posizioni dei due bit meno significativi.

Problema Campione:

(40) = 5D
Risultato = (41) = 74

4) Preleva i Quattro Bit Più Significativi

Scopo: Posiziona i quattro bit più significativi dei contenuti della locazione di memoria 40 nella locazione di memoria 41. Azzera i quattro bit meno significativi della locazione 41.

Problema Campione:

(40) = C4
Risultato = (41) = C0

5) Pone una locazione di Memoria a Tutti Uni

Scopo: La locazione di Memoria 40 è posto a tutti uni (FF Esadec.).

6) Assemblaggio di una Parola

Scopo: Combina i quattro bit meno significativi delle locazioni di memoria 40 e 41 in una parola e l'immagazzina nella locazione di memoria 42. Posiziona i quattro bit meno si-

gnificativi della locazione di memoria 40 nei quattro bit più significativi della locazione 42; posiziona i quattro bit meno significativi della locazione 41 nei quattro meno significativi della locazione 42.

Problema Campione:

(40)	=	6A
(41)	=	B3
Risultato	=	(42) = A3

7) Ricerca del Più Piccolo di Due Numeri

Scopo: Posiziona il minore dei contenuti delle locazioni di memoria 40 e 41 nella locazione 42. Si assume che le locazioni 40 e 41 tacitamente contengano numeri binari.

Problemi Campione:

a.	(40)	=	3F
	(41)	=	2B
	Risultato	=	(42) = 2B
b.	(40)	=	75
	(41)	=	A8
	Risultato	=	(42) = 75

8) Addizione a 24 Bit

Scopo: Somma il numero di 24 bit delle locazioni di memoria 40, 41 e 42 al numero di 24 bit delle locazioni 43, 44 e 45. Gli otto bit più significativi sono nelle locazioni di memoria 42 e 45, gli otto meno significativi nelle 40 e 43. Immagazzina il risultato nelle locazioni di memoria 46, 47 e 48 con i bit più significativi in 48 e quelli meno significativi in 46.

Problema Campione:

(40)	=	2A
(41)	=	67
(42)	=	35
(43)	=	F8
(44)	=	A4
(45)	=	51
Risultato	=	(46) = 22
		(47) = 0C
		(48) = 87
cioè:		35672A
	+	51A4F8
		870C22

9) Somma di Quadrati

Scopo: Calcola i quadrati dei contenuti delle locazioni di memoria 40 e 41 e li somma assieme. Posiziona il risultato nella locazione di memoria 42. Si assuma che le locazioni 40 e 41 contengano un numero da 0 a 7 compresi cioè $0 \leq (40) \leq 7$ e $0 \leq (41) \leq 7$. Si usi la tabella dei quadrati dall'esempio intitolato «Tabella dei Quadrati».

Problema Campione:

(40) = 03
(41) = 06
Risultato = (42) = 2D
cioè $3^2 + 6^2 = 9 + 36$ (decimale)
$= 45 =$ 2D (esadec.)

10) Complemento a Due di 16 Bit

Scopo: Posiziona il complemento a due di un numero a 16 bit delle locazioni 40 e 41 (bit più significativi in 41) nelle locazioni di memoria 42 e 43 (bit più significativi in 43).

Problema Campione:

(40) = 00
(41) = 58
Risultato = (42) = 00
(43) = A8

Capitolo 5
SEMPLICI CICLI DI PROGRAMMA

Il ciclo di programma è la struttura base che forza la CPU a ripetere una sequenza di istruzioni. I cicli si dividono in quattro parti:

1) La INIZIALIZZAZIONE che stabilisce il valore di partenza del contatore, i Registri di Indirizzo (Puntatori) ed altre variabili.
2) Il CORPO dove ha sede la manipolazione dei dati attuali. Questa è la parte che esegue il lavoro.
3) La parte CONTROLLO del ciclo che aggiorna i contatori ed i puntatori per la successiva iterazione.
4) La parte FINALE che analizza ed immagazzina i risultati.

Si noti che il computer esegue le parti 1 e 4 solo una volta mentre può eseguire le parti 2 e 3 molte volte. Così il tempo di esecuzione di un ciclo sarà direttamente dipendente dal tempo di esecuzione delle parti 2 e 3. Si richiederà che le parti 2 e 3 vengano eseguite più velocemente possibile e non ci si preoccuperà del tempo di esecuzione delle parti 1 e 4. Un tipico ciclo di un programma può essere rappresentato dal diagramma di flusso di Figura 5-1 ovvero le posizioni del corpo e del controllo possono essere invertite come indicato in Figura 5-2. La parte di elaborazione (corpo) della Figura 5-1 è sempre eseguita almeno una volta mentre il corpo della Figura 5-2 può non essere mai eseguito. La Figura 5-1 sembra più naturale ma la Figura 5-2 è spesso più efficiente ed evita il problema della non esistenza di dati (che è il punto debole per un computer e causa frequente di istruzioni banali come sollecitare qualcuno per un conto di $ 0.00).

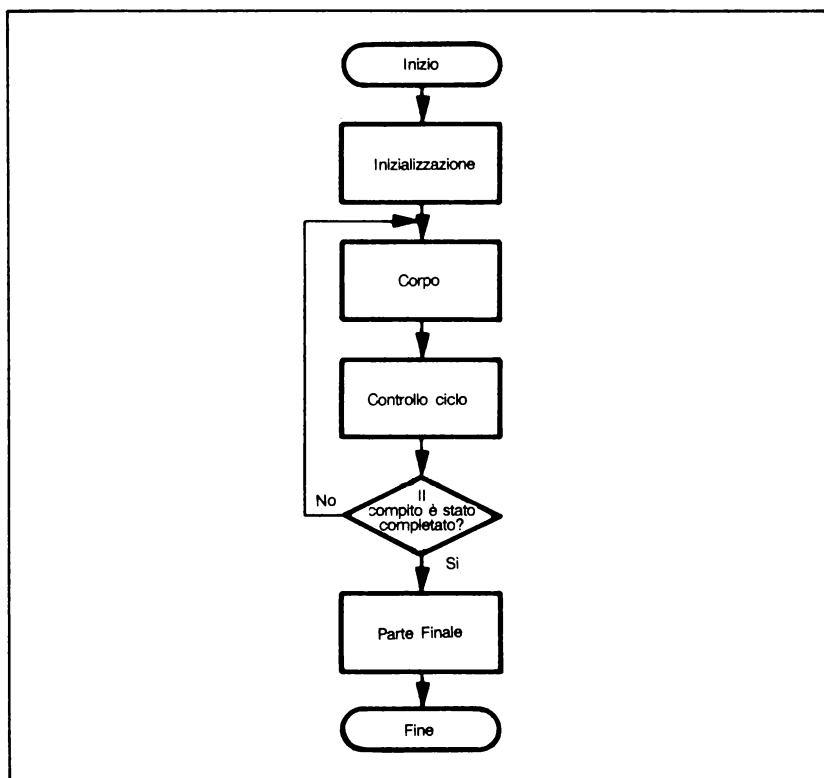



Figura 5-1
Diagramma Di Flusso Di Un Ciclo Di Programma

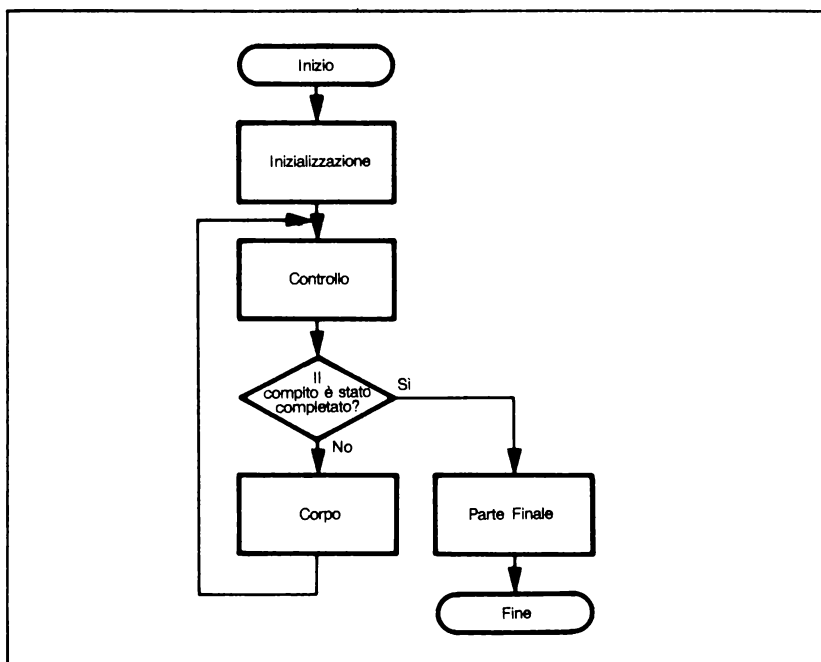



Figura 5-2
Un Ciclo Di Programma Che Consente Zero Iterazioni

La struttura del ciclo può elaborare interi blocchi di dati. Per effettuare questo il programma deve incrementare un registro Indirizzo (normalmente la coppia di registri H) dopo ogni iterazione cosicchè il registro Indirizzo punta l'elemento successivo del blocco dati. La successiva iterazione sarà poi eseguita con le stesse operazioni sui dati della successiva locazione di memoria. Il computer può trattare blocchi di qualsiasi lunghezza con lo stesso set d'istruzioni. Il registro M è la chiave del corpo del ciclo con l'8080, poichè esso permette di variare l'indirizzo di memoria attuale cambiando il contenuto della coppia di registri H.

ESEMPI

Somma di Dati

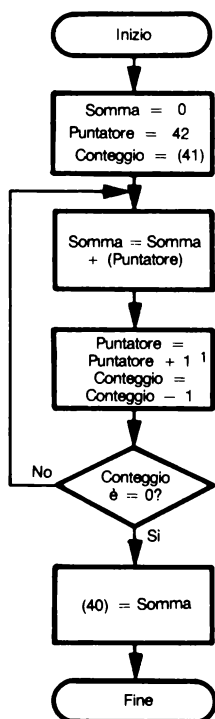
Scopo: Calcola la somma di una serie di numeri. La lunghezza della serie si trova nella locazione di memoria 41 e la serie stessa inizia dalla locazione di memoria 42. Immagazzina la somma nella locazione 40. Si assume che la somma sia un numero ad 8 bit cosicchè è possibile ignorare i riporti.

SOMMA AD 8 BIT

Problema Campione:

(41)	=	03
(42)	=	28
(43)	=	55
(44)	=	26
Risultato	=	(42) + (43) + (44) = (40)
		28 + 55 + 26 = A3

Ci sono tre addendi della somma poichè (41) = 03

Diagramma di Flusso:

Programma Sorgente:

```
        LXI     H,41H
        MOV     B,M     ;CONTEGGIO = LUNGHEZZA DELLA SERIE DI NUMERI
        SUB     A       ;SOMMA = 0
SUMD:   INX     H
        ADD     M       ;SOMMA = SOMMA + DATI
        DCR     B
        JNZ     SUMD
        STA     40H     ;IMMAGAZZINA SOMMA
HERE:   JMP     HERE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)	Contenuti di Mem. (Esadec.)
00	LXI H,41H	21
01		41
02		00
03	MOV B,M	46
04	SUB A	97
05	SUMD: INX H	23
06	ADD M	86
07	DCR B	05
08	JNZ SUMD	C2
09		05
0A		00
0B	STA 40H	32
0C		40
0D		00
0E	HERE: JMP HERE	C3
0F		0E
10		00

Le prime tre istruzioni costituiscono la parte di inizializzazione del programma. Esse pongono il puntatore, il contatore e la somma ai loro valori di partenza. Si noti che MOV può trasferire dati tra la memoria e qualunque registro per Scopi Generali, mentre LDA e STA possono trasferire dati solo tra la memoria e l'Accumulatore.

Il corpo del ciclo è la semplice istruzione ADDM che somma i contenuti della locazione di memoria indirizzata dai Registri H ed L ai contenuti dell'Accumulatore ed immagazzina il risultato nell'Accumulatore. Questa istruzione esegue il vero compito del programma.

La parte di controllo del ciclo consiste delle istruzioni INX H e DCR B. INX H aggiorna il puntatore in modo che la successiva iterazione somma il numero successivo alla somma. DCR B decrementa il contatore da cui risulta il numero di iterazioni eseguite.

L'istruzione JNZ SUMD causa un salto alla locazione SUMD se il flag Zero è zero. Se il flag Zero è uno la CPU esegue l'istruzione sequenzialmente successiva (cioè STA 40H). Poichè DCR B è l'ultima istruzione, prima di JNZ, ad influenzare il flag Zero, JNZ SUMD origina un Salto a SUMD se BCR B non produce un risultato zero, cioè:

$$(PC) = SUMD \text{ se } (B) \neq 0$$
$$(PC) = (PC) + 3 \text{ se } (B) = 0$$

(PC viene incrementato di 3 poichè l'istruzione JNZ è formata da tre parole). Una singola istru-

zione che combina il Decremento ed il Salto sarebbe una pratica aggiunta al set d'istruzioni dell'8080.

La maggior parte dei cicli dei computer decrementano piuttosto che aumentare un contatore cosicchè è possibile usare il Flag Zero come condizione di uscita dal ciclo. Si ricordi che il flag Zero è 1 se il risultato era 0 e 0 se il risultato era non zero. Si provi a riscrivere il programma nel caso che il contatore venga aumentato piuttosto che decrementato; quale metodo è più efficiente.

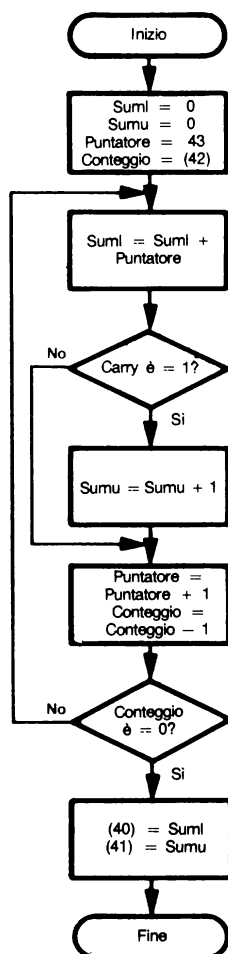
L'ordine delle istruzioni è spesso molto importante. Per esempio DCR B deve essere eseguita proprio prima di JNZ SUMD poichè diversamente il risultato del flag zero posto da DCR B potrebbe essere cambiato da un'altra istruzione. Inoltre INX H deve essere eseguita prima di ADD M altrimenti il primo numero sommato alla somma sarà (41) anzichè (42).

Somma di Dati a 16 Bit

Scopo: Calcola la somma di una serie di numeri. La lunghezza della serie è specificata dalla locazione 42, e la serie stessa inizia dalla locazione di memoria 43. Immagazzina la somma nelle locazioni di memoria 40 e 41 (con gli otto bit più significativi in 41).

Problema Campione:

(42) = 03
(43) = C8
(44) = FA
(45) = 96
Risultato = C8 + FA + 96 = 0258 (Esadec.)
(40) = 58
(41) = 02

Diagramma di Flusso:

Programma Sorgente:

```
        LXI     H,42H
        MOV     B,M       ;CONTEGGIO = LUNGHEZZA DELLA SERIE
        SUB     A         ;BIT MENO SIGNIFICATIVI DI SUM = 0
        MOV     C,A       ;BIT PIÙ SIGNIFICATIVI DI SUM = 0
DSUMD:  INX     H
        ADD     M         ;SUM = SUM + DATI
        JNC     CHCNT
        JNR     C         ;SOMMA CARRY AI BIT PIÙ SIGNIFICATIVI DI SUM
CHCNT:  DCR     B
        JNZ     DSUMD
        LXI     H,40H     ;IMMAGAZZINA I BIT MENO SIGNIFICATIVI DI SUM
        MOV     M,A
        INX     H         ;IMMAGAZZINA I BIT PIÙ SIGNIFICATIVI DI SUM
        MOV     M,C
        HERE:   JMP       HERE
```

Programma Oggetto:

Contenuti di Mem. (Esadec.)	Istruzione (Mnemonico)	Indirizzo di Mem. (Esadec.)
00	LXI H,42H	21
01		42
02		00
03	MOV B,M	46
04	SUB A	97
05	MOV C,A	4F
06	DSUMD: INX H	23
07	ADD M	86
08	JNC CHCNT	D2
09		0C
0A		00
0B	INR C	0C
0C	CHCNT: DCR B	05
0D	JNZ CHCNT	C2
0E		OC
0F		00
10	LXI H,40H	21
11		40
12		00
13	MOV M,A	77
14	INX H	23
15	MOV M,C	71
16	HERE: JMP HERE	C3
17		16
18		00

La struttura di questo programma è la stessa del precedente. I bit più significativi della somma ora devono essere inizializzati ed immagazzinati. Il corpo del ciclo consiste di tre istruzioni (ADD M, JNC CHCNT, INR C) e comprende un Salto condizionato.

JNC CHCNT origina un salto alla locazione di memoria CHCNT se Carry = 0. Cosi se non c'è Carry dall'addizione ad 8 bit il programma salta verso lo statement che incrementa i bit più significativi della somma.

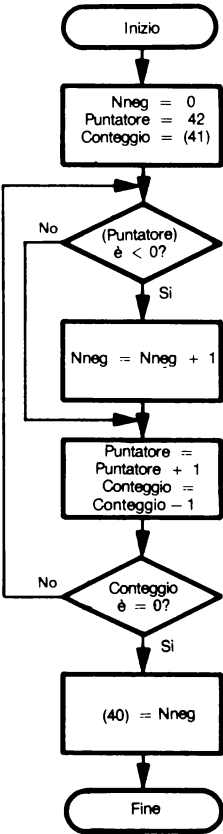
INR C somma 1 ai contenuti del registro C. INR è un incremento ad 8 bit; INX è un incremento a 16 bit.

Numero di elementi negativi

Scopo: Determina il numero di elementi negativi (bit significativo uguale ad 1) in un blocco di dati. La lunghezza del blocco si trova indicata nella locazione di memoria 41 ed il blocco stesso inizia dalla locazione di memoria 42. Immagazzina il numero di elementi negativi nella locazione di memoria 40.

Problema Campione:

(41) = 06
(42) = 68
(43) = F2
(44) = 87
(45) = 30
(46) = 59
(47) = 2A
Risultato = (40) = 02 poichè 43 e 44 contengono numeri con il bit più significativo uguale ad 1

Diagramma di Flusso:

Programma Sorgente:

```
        LXI     H,41H
        MOV     B,M         ;CONTEGGIO = NUMERO DI ELEMENTI
        MVI     A,01111111B ;NUMERO POSITIVO PIÙ GRANDE PER PARAGONE
        MVI     C,0         ;NUMERO DI NEGATIVI = 0
SRNEG:  INX     H
        CMP     M           ;IL BIT PÙ SIGNIFICATIVO DEI DATI È = 1?
        JNC     CHCNT       ;NO SE NON È RICHIESTO PRESTITO
        INR     C           ;SÌ, SOMMA L AL NUMERO DI NEGATIVI
CHCNT:  DCR     B
        JNZ     SRNEG
        MOV     A,C
        STA     40H         ;IMMAGAZZINA IL NUMERO DI NEGATIVI
HERE:   JMP     HERE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)	Contenuti di Mem. (Esadec.)
00	LXI H,41H	21
01		41
02		00
03	MOV B,M	46
04	MVI A,01111111B	3E
05		7F
06	MVI C,0	0E
07		00
08	SRNEG: INX H	23
09	CMP M	BE
0A	JNC CHCNT	D2
0B		0E
0C		00
0D	INR C	0C
0E	CHCNT: DCR B	05
0F	JNZ SRNEG	C2
10		08
11		00
12	MOV A,C	79
13	STA 40H	32
14		40
15		00
16	HERE: JMP HERE	C3
17		16
18		00

Tutti i numeri negativi hanno il bit più significativo uguale ad 1. Così i numeri negativi sono attualmente più larghi di quelli positivi quando si usa la notazione complemento a due. CMP M posiziona i flag come se i contenuti della locazione di memoria indirizzata dai Registri H ed L fossero stati sottratti dai contenuti dell'Accumulatore. Né l'Accumulatore nè la locazione di memoria sono però stati variati. Così il valore dell'Accumulatore viene conservato per la successiva iterazione.

CMP M pone i flag come segue:

Zero	$= 1$ se $(A) = (M)$
	$= 0$ se $(A) \neq (M)$
Carry	$= 1$ se $(A) < (M)$
	$= 0$ se $(A) \geq (M)$, cioè la sottrazione non richiede un Prestito

Poichè A contiene 7F (Esadec.) nel programma i flag saranno:

Zero	$= 1$ se $(M) = 7F$
	$= 0$ se $(M) \neq 7F$
Carry	$= 1$ se $(M) > 7F$
	$= 0$ se $(M) \leq 7F$

Così se (M) ha il bit più significativo uguale ad 1, il Carry sarà 1; se (M) ha il bit più significativo uguale a 0, il Carry sarà 0.

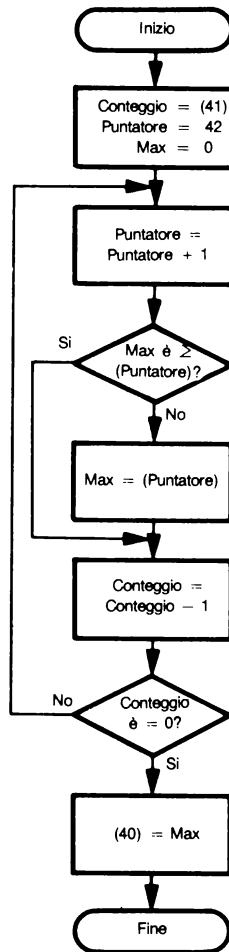
L'istruzione CMP è necessaria per posizionare i flag. Il posizionamento dei dati nell'Accumulatore (MOV A,M) non influenza i flag. Si potrebbe usare ORA A oppure ANA A che influenzano i flag ma non cambiano il valore nell'Accumulatore. CMP è conveniente perchè influenza i flag senza cambiare nessun registro e può cosi essere usata direttamente per esaminare i contenuti di una locazione di memoria. L'istruzione CMP influenza anche il flag Sign. Se si sostituisce l'istruzione JNC CHCNT con JM CHCNT è possibile ristrutturare la parte di inizializzazione del programma sorgente cosicchè esso esegua il compito specifico?

Ricerca del Massimo

Scopo: Trova l'elemento maggiore di un blocco di dati. La lunghezza del blocco è indicata nella locazione di memoria 41 ed il blocco stesso inizia dalla locazione di memoria 42. Immagazzina il massimo nella locazione di memoria 40. Si assume tacitamente che i numeri del blocco sono tutti binari ad 8 bit.

Problema Campione:

(41)	=	05
(42)	=	67
(43)	=	79
(44)	=	15
(45)	=	E3
(46)	=	72
Risultato	=	(40) = E3 poichè questo è il maggiore dei cinque numeri precedenti

Diagramma di flusso:

Programma Sorgente:

```
        LXI   H,41H    ;PUNTA A CONTEGGIO
        MOV   B,M      ;CONTEGGIO = NUMERO DI ELEMENTI
        SUB   A        ;MASSIMO = MINIMO VALORE POSSIBILE (ZERO)
NEXTE:  INX   H
        CMP   M        ;IL SUCCESSIVO ELEMENTO È > MASSIMO?
        JNC   DECNT
        MOV   A,M      ;SÌ, AGGIORNA IL MASSIMO
DECNT:  DCR   B
        JNZ   NEXTE
        STA   40H      ;CONSERVA IL MASSIMO
HERE:   JMP   HERE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)	Contenuti di Mem. (Esadec.)
00	LXI H,40H	21
01		40
02		00
03	MOV B,M	46
04	SUB A	97
05	NEXTE: INX H	23
06	CMP M	BE
07	JNC DECNT	D2
08		0B
09		00
0A	MOV A,M	7E
0B	DECNT: DCR B	05
0C	JNZ NEXTE	C2
0D		05
0E		00
0F	STA 40H	32
10		40
11		00
12	HERE: JMP HERE	C3
13		12
14		00

I primi tre byte di questo programma formano la parte di inizializzazione. L'istruzione SUB A azzera l'Accumulatore.

Questo programma si basa sul fatto che zero è il più piccolo numero binario ad 8 bit. Quindi si predispone il registro che conterrà il massimo, in questo caso l'Accumulatore, al minimo valore possibile prima di entrare nel ciclo. Allora il programma predisporrà l'Accumulatore al valore massimo a meno che tutti gli elementi della lista siano uguali a zero.

Il programma funziona correttamente se c'è un solo elemento ma non se non ne esiste nessuno — Perchè? Come si potrebbe risolvere questo problema?

L'itruzione CMP M pone il flag Carry come segue:

$$\text{Carry} = 1 \text{ se l'ELEMENTO} > \text{MAX}$$
$$= 0 \text{ se l'ELEMENTO} \leq \text{MAX}$$

Se Carry = 0 il programma procede a DECNT e non cambia il massimo. Se Carry = 1 il programma sostituisce il vecchio massimo con l'elemento attuale.
Il programma non funziona se si hanno numeri con segno perchè i numeri negativi appariranno più grandi di quelli positivi. Il problema viene talvolta complicato da overflow; si può rimediare ricordando che nella sottrazione si ha overflow solo se i numeri hanno segno opposto. Perchè? Se i numeri hanno segno opposto chiaramente il numero positivo è il maggiore. La soluzione potrebbe essere più semplice se l'8080 avesse un bit di overflow.

Aggiusta una Frazione Binaria

Scopo: Sposta a sinistra i contenuti della locazione di memoria 40 fino a che il bit più significativo del numero è 1. Immagazzina il risultato nella locazione di memoria 41 ed il numero di spostamenti a sinistra richiesti nella locazione di memoria 42. Se i contenuti della locazione di memoria 40 sono zero, azzera sia 41 che 42.

Si noti questo equivale alla conversione di un numero in notazione scientifica, per esempio 0,0057 = 5,7 x 10^3.

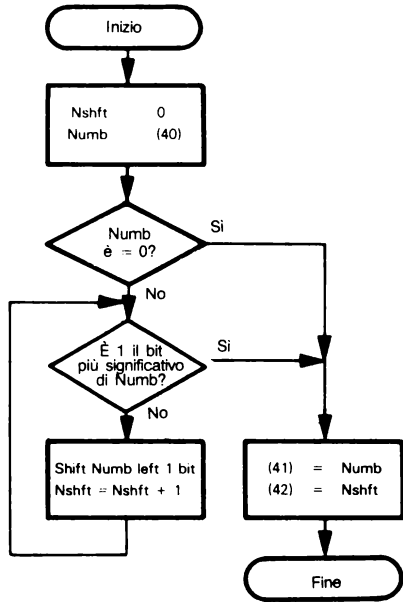
Problemi Campione:

a. (40) = 22
Risultato = (41) = 88
(42) = 02

b. 40 = 01
Risultato = (41) = 80
(42) = 07

c. (40) = CB
Risultato = (41) = CB
(42) = 00

d. (40) = 00
Risultato = (41) = 00
(42) = 00

Diagramma di flusso:

Programma Sorgente:

```
        SUB   A
        MOV   B,A     ;NUMERO DI SPOSTAMENTI = 0
        LXI   H,40H
        ADD   M       ;ACCETTA I DATI
        JZ    DONE    ;SALTA SE I DATI SONO = 0
CHKMS:  JM    DONE    ;IL PROSSIMO BIT È 1?
        JNR   B       ;NO, SOMMA 1 AL NUMERO DI SPOSTAMENTI
        ADD   A       ;SPOSTA A SINISTRA DI 1 BIT
        JMP   CHKMS
DONE:   INX   H       ;CONSERVA I DATI AGGIUSTATI
        MOV   M,A
        INX   H
        MOV   M,B     ;CONSERVA IL NUMERO DI SPOSTAMENTI
HERE:   JMP   HERE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		SUB	A	97
01		MOV	B,A	47
02		LXI	H,40H	21
03				40
04				00
05		ADD	M	86
06		JZ	DONE	CA
07				11
08				00
09	CHKMS:	JM	DONE	FA
0A				11
0B				00
0C		INR	B	04
0D		ADD	A	87
0E		JMP	CHKMS	C3
0F				09
10				00
11	DONE:	INX	H	23
12		MOV	M,A	77
13		INX	H	23
14		MOV	M,B	70
15	HERE:	JMP	HERE	C3
16				16
17				00

L'istruzione JM DONE origina un Salto alla locazione DONE se il bit Sign è uguale ad 1. Questa condizione può significare che l'ultimo risultato era un numero negativo oppure può significare che il bit più significativo era 1 – il computer fornisce il risultato; il programmatore deve provvedere all'interpretazione.

ADD A somma il numero dell'Accumulatore a sè stesso. Il programma usa questa istruzione piuttosto che RAL od RLC perchè ADD A influenza il bit Sign mentre questo non avviene per RAL od RLC.

PROBLEMI

1) Or-Esclusivo di un Blocco di Dati (Checksum)

Scopo: Calcola l'OR-Esclusivo di una serie di numeri. La lunghezza della serie è espressa nella locazione di memoria 41 e la serie stessa inizia dalla locazione di memoria 42. Immagazzina l'OR-Esclusivo (checksum) nella locazione di memoria 40. Il risultato dell'operazione di OR-Esclusivo di tutti i numeri della serie viene formato nell'Accumulatore.

Si noti che questa operazione viene spesso usata nel nastro perforato e nei sistemi a cassette per assicurare che i dati siano dati letti correttamente. L'OR-ESCLUSIVO calcolato è paragonato ad un valore memorizzato con i dati — se i due numeri non sono in accordo il sistema normalmente indicherà un errore all'operatore ed automaticamente leggerà ancora i dati.

Problema Campione:

(41) = 03
(42) = 28
(43) = 55
(44) = 26

$$\text{Risultato} = (42) \oplus (43) \oplus (44) = 28 \oplus 55 \oplus 26$$

$$\begin{array}{rl} = & 00101000 \\ \oplus & \underline{01010101} \\ & 01111101 \\ \oplus & \underline{00100110} \\ & 01011011 \\ = & 5B \end{array}$$

2) Somma di Dati a 16 Bit

Scopo: Calcola la somma di una serie di numeri a 16 bit. La lunghezza della serie è espressa nella locazione di memoria 42 e la serie stessa inizia dalla locazione di memoria 43. Immagazzina la somma nelle locazioni di memoria 40 e 41 (gli otto bit più significativi si trovano in 41). Ogni numero a 16 bit occupa due locazioni di memoria con gli 8 bit più significativi nella seconda. Si suppone che la somma possa essere contenuta in 16 bit (cioè senza Riporti).

Problema Campione:

(42) = 03
(43) = F1
(44) = 28
(45) = 1A
(46) = 30
(47) = 89
(48) = 4B

Risultato = 28F1 + 301A + 4B89 = A494
(40) = 94
(41) = A4

3) Numero di Numeri Positivi, Negativi e Nulli

Scopo: Determina il numero di numeri positivi (bit più significativi uguale a zero ma l'intero numero non zero), negativi (bit più significativo uguale ad 1) o nulli di un insieme. La lunghezza del blocco viene indicato nella locazione di memoria 43 ed il blocco stesso inizia dalla locazione 44. Posiziona il numero di elementi negativi nella locazione 40, il numero di elementi zero nella locazione 41 ed il numero di elementi positivi nella locazione 42.

Problema Campione:

(43) = 06
(44) = 68
(45) = F2
(46) = 87
(47) = 00
(48) = 59
(49) = 2A
Risultato = 2 negativi, 3 positivi ed 1 solo zero
(40) = 02
(41) = 01
(42) = 03

4) Ricerca del minimo

Scopo: Trova l'elemento più piccolo di un blocco dati. La lunghezza del blocco si trova nella locazione di memoria 41 e lo stesso blocco inizia dalla locazione 42. Immagazzina il minimo nella locazione 40. Si suppone tacitamente che i numeri del blocco sono binari ad 8 bit.

Problema Campione:

(41) = 05
(42) = 67
(43) = 79
(44) = 15
(45) = E3
(46) = 72
Risultato = (40) = 15 poichè questo è l'elemento più piccolo
dei cinque numeri precedenti

5) Conteggio dei Bit 1

Scopo: Posiziona il numero di bit 1 dei contenuti della locazione di memoria 40 nella locazione 41.

Problema Campione:

(40) = 3B = 00111011
Risultato = (41) = 05

Capitolo 6

DATI ESPRESSI IN CARATTERI CODIFICATI

I microprocessori spesso trattano dati espressi in caratteri codificati. Non solo le tastiere, le telescriventi, i dispositivi di comunicazione, i display ed i terminali di computer ricevono ed emettono dati in caratteri codificati – ma anche molti strumenti e sistemi di controllo lo fanno. Il codice più comunemente usato è l'ASCII. I codici Baudot ed EBCDIC vengono impiegati meno frequentemente. Si assumerà che tutti i dati espressi in caratteri codificati siano ASCII a 7 bit con il bit più significativo uguale a zero (Vedere Tabella 6-1).

Alcuni artifici per ricordare i metodi di codificazione ASCII dei dati sono:

ELABORAZIONE DATI IN ASCII

1) Il codice per i numeri e le lettere forma sequenze ordinate. I numeri decimali vanno da 30 a 39 espressi in esadecimali, le lettere maiuscole da 41 a 5A espresse in esadecimale. Così è possibile fare un elenco alfabetico selezionando i dati in ordine numerico crescente.
2) Il computer non riscontra distinzioni tra caratteri tipografici e non. Queste distinzioni vengono fatte solo dati dispositivo I/O.
3) Un dispositivo ASCII interpreterà ed invierà tutti i dati come ASCII. Per stampare un 7 con una stampante ASCII il microprocessore deve inviare l'esadecimale 37; l'esadecimale 07 è il carattere 'campanello'. Analogamente il microprocessore riceverà il carattere 9 da una tastiera ASCII come esadecimale 39; l'esadecimale 09 è il carattere 'tab'.
4) Alcuni dispositivi ASCII non usano l'intero set di caratteri. Per esempio i caratteri di controllo e le lettere minuscole possono essere ignorati oppure stampati come '?'.
5) Alcuni ASCII largamente impiegati sono:
   OA - avanzamento linea (LF)
   OD - ritorno carrello (CR)
   20 - spazio
   3F - ?
   7F - rubout o cancella carattere

Tabella 6-1
Tabella Dei Codici Dei Caratteri ASCII A 7-Bit

<table>
<tr><th>Esadec.</th><th>ASCII</th><th>Esadec.</th><th>ASCII</th><th>Esadec.</th><th>ASCII</th></tr>
<tr><td>00</td><td>NUL</td><td>2B</td><td>+</td><td>56</td><td>V</td></tr>
<tr><td>01</td><td>SOH</td><td>2C</td><td>,</td><td>57</td><td>W</td></tr>
<tr><td>02</td><td>STX</td><td>2D</td><td>—</td><td>58</td><td>X</td></tr>
<tr><td>03</td><td>ETX</td><td>2E</td><td>.</td><td>59</td><td>Y</td></tr>
<tr><td>04</td><td>EOT</td><td>2F</td><td>/</td><td>5A</td><td>Z</td></tr>
<tr><td>05</td><td>ENQ</td><td>30</td><td>0</td><td>5B</td><td>[</td></tr>
<tr><td>06</td><td>ACK</td><td>31</td><td>1</td><td>5C</td><td>\</td></tr>
<tr><td>07</td><td>BEL</td><td>32</td><td>2</td><td>5D</td><td>]</td></tr>
<tr><td>08</td><td>BS</td><td>33</td><td>3</td><td>5E</td><td>∧ (↑)</td></tr>
<tr><td>09</td><td>HT</td><td>34</td><td>4</td><td>5F</td><td>— (←)</td></tr>
<tr><td>0A</td><td>LF</td><td>35</td><td>5</td><td>60</td><td>`</td></tr>
<tr><td>0B</td><td>VT</td><td>36</td><td>6</td><td>61</td><td>a</td></tr>
<tr><td>0C</td><td>FF</td><td>37</td><td>7</td><td>62</td><td>b</td></tr>
<tr><td>0D</td><td>CR</td><td>38</td><td>8</td><td>63</td><td>c</td></tr>
<tr><td>0E</td><td>SO</td><td>39</td><td>9</td><td>64</td><td>d</td></tr>
<tr><td>0F</td><td>SI</td><td>3A</td><td>:</td><td>65</td><td>e</td></tr>
<tr><td>10</td><td>DLE</td><td>3B</td><td>;</td><td>66</td><td>f</td></tr>
<tr><td>11</td><td>DC1 (X-ON)</td><td>3C</td><td><</td><td>67</td><td>g</td></tr>
<tr><td>12</td><td>DC2 (TAPE)</td><td>3D</td><td>=</td><td>68</td><td>h</td></tr>
<tr><td>13</td><td>DC3 (X-OFF)</td><td>3E</td><td>></td><td>69</td><td>i</td></tr>
<tr><td>14</td><td>DC4 (~~TAPE~~)</td><td>3F</td><td>?</td><td>6A</td><td>j</td></tr>
<tr><td>15</td><td>NAK</td><td>40</td><td>"</td><td>6B</td><td>k</td></tr>
<tr><td>16</td><td>SYN</td><td>41</td><td>A</td><td>6C</td><td>l</td></tr>
<tr><td>17</td><td>ETB</td><td>42</td><td>B</td><td>6D</td><td>m</td></tr>
<tr><td>18</td><td>CAN</td><td>43</td><td>C</td><td>6E</td><td>n</td></tr>
<tr><td>19</td><td>EM</td><td>44</td><td>D</td><td>6F</td><td>o</td></tr>
<tr><td>1A</td><td>SUB</td><td>45</td><td>E</td><td>70</td><td>p</td></tr>
<tr><td>1B</td><td>ESC</td><td>46</td><td>F</td><td>71</td><td>q</td></tr>
<tr><td>1C</td><td>FS</td><td>47</td><td>G</td><td>72</td><td>r</td></tr>
<tr><td>1D</td><td>GS</td><td>48</td><td>H</td><td>73</td><td>s</td></tr>
<tr><td>1E</td><td>RS</td><td>49</td><td>I</td><td>74</td><td>t</td></tr>
<tr><td>1F</td><td>US</td><td>4A</td><td>J</td><td>75</td><td>u</td></tr>
<tr><td>20</td><td>SP</td><td>4B</td><td>K</td><td>76</td><td>v</td></tr>
<tr><td>21</td><td>!</td><td>4C</td><td>L</td><td>77</td><td>w</td></tr>
<tr><td>22</td><td>"</td><td>4D</td><td>M</td><td>78</td><td>x</td></tr>
<tr><td>23</td><td>#</td><td>4E</td><td>N</td><td>79</td><td>y</td></tr>
<tr><td>24</td><td>$</td><td>4F</td><td>O</td><td>7A</td><td>z</td></tr>
<tr><td>25</td><td>%</td><td>50</td><td>P</td><td>7B</td><td>¦</td></tr>
<tr><td>26</td><td>&</td><td>51</td><td>Q</td><td>7C</td><td>|</td></tr>
<tr><td>27</td><td>'</td><td>52</td><td>R</td><td>7D</td><td>¦ (ALT MODE)</td></tr>
<tr><td>28</td><td>(</td><td>53</td><td>S</td><td>7E</td><td>~</td></tr>
<tr><td>29</td><td>)</td><td>54</td><td>T</td><td>7F</td><td>DEL (RUB OUT)</td></tr>
<tr><td>2A</td><td>*</td><td>55</td><td>U</td><td></td><td></td></tr>
</table>

ESEMPI

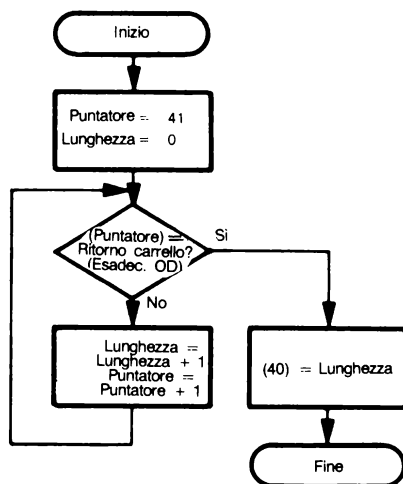
Lunghezza di una Stringa di Caratteri

Scopo: Determina la lunghezza di una stringa di caratteri ASCII (7 bit con quello più significativo uguale a 0). La stringa inizia dalla locazione di memoria 41; la fine della stringa è segnata da un carattere di ritorno carrello. ('CR', esadecimale OD). Posiziona la lunghezza della stringa (escluso il ritorno carrello) nella locazione di memoria 40.

Problemi Campione:

a. (40) = OD
Risultato = (40) = 00 poichè il primo carattere è un ritorno carrello

b. (41) = 52 R
(42) = 41 A
(43) = 54 T
(44) = 48 H
(45) = 45 E
(46) = 52 R
(47) = OD 'CR'
Risultato = (40) = 06

Diagramma di Flusso:

Programma Sorgente:

```
        LXI     H,41H    ;PUNTATORE = INIZIO DELLA STRINGA
        MVI     B,0      ;LUNGHEZZA = 0
        MVI     A,ODH
CHKCR   CMP     M        ;È IL CARATTERE 'CR'?
        JZ      DONE     ;SÌ, FINE DELLA STRINGA
        INR     B        ;NO, AGGIUNGI 1 ALLA LUNGHEZZA
        INX     H
        JMP     CHKCR    ;ESAMINA IL CARATTERE SUCCESSIVO
DONE:   MOV     A,B
        STA     40H      ;CONSERVA LA LUNGHEZZA DELLA STRINGA
HERE:   JMP     HERE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		LXI	H,41H	21
01				41
02				00
03		MVI	B,0	06
04				00
05		MVI	A,0DH	3E
06				0D
07	CHKCR:	CMP	M	BE
08		JZ	DONE	CA
09				10
0A				00
0B		INR	B	04
0C		INX	H	23
0D		JMP	CHKCR	C3
0E				07
0F				00
10	DONE:	MOV	A,B	78
11		STA	40H	32
12				40
13				00
14	HERE:	JMP	HERE	C3
15				14
16				00

Il ritorno carrello ('CR', 0D) è anch'esso un carattere ASCII per come è concepito il computer. Il fatto che il dispositivo I/O tratta 'CR' come un carattere di controllo piuttosto che un carattere tipografico non influenza il computer.

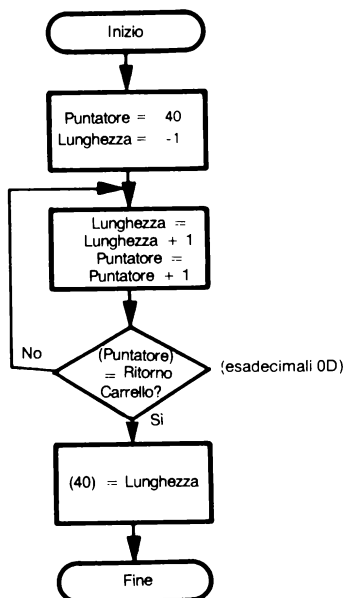
L'istruzione Paragona (CMP) posiziona i flag ma consente al carattere di ritorno carrello di rimanere nell'Accumulatore per ulteriori paragoni. Il flag Zero viene posizionato come segue:

Zero = 1 se il carattere della stringa è un ritorno carrello
Zero = 0 se non è un ritorno carrello

L'istruzione INR B somma 1 al contatore della lunghezza della stringa nel Registro B. MVI B,0 inizializza a zero il contatore prima del ciclo. Si ricordi di inizializzare le variabili prima di usare in un ciclo. Questo ciclo non termina fino a che il contatore non è decrementato a zero.

Il computer continuerà semplicemente nell'esame dei caratteri fino a che non trova un ritorno carrello. Si può predisporre un massimo conteggio in un ciclo in modo da evitare problemi con stringhe non corrette che non contengono un ritorno al carrello. Cosa succederebbe se la stringa non contenesse un ritorno carrello?

Si noti che riordinando la logica e cambiando le condizioni iniziali si possono eliminare diversi byte di codice oggetto e diminuire il tempo di esecuzione del ciclo. Se si aggiusta il diagramma di flusso in modo che il programma incrementi il contatore ed il puntatore prima della ricerca del ritorno carrello, è necessaria una sola istruzione di salto al posto di due. I nuovi diagramma di flusso e programma sono:

Diagramma di flusso:

Programma Sorgente:

```
        LXI     H,40H    ;PUNTATORE = BYTE PRIMA DELLA STRINGA
        MVI     B,0FFH   ;LUNGHEZZA = -1
        MVI     A,0DH
CHKCR:  INX     H
        INR     B        ;SOMMA 1 ALLA LUNGHEZZA
        CMP     M        ;È IL CARATTERE 'CR'?
        JNZ     CHKCR    ;NO, GESTISCI IL CONTEGGIO
        MOV     A,B
        STA     40H      ;SÌ, CONSERVA LA LUNGHEZZA DELLA STRINGA
HERE:   JMP     HERE
```

Programma Oggetto

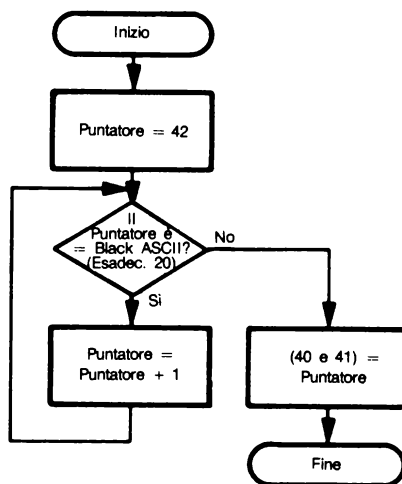
Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		LXI	H,40H	21
01				40
02				00
03		MVI	B,0FFH	06
04				FF
05		MVI	A,0DH	3E
06				0D
07	NEXT:	INX	H	23
08		INR	B	04
09		CMP	M	BE
0A		JNZ	CHKCR	C2
0B				07
0C				00
0D		MOV	A,B	78
0E		STA	40H	32
0F				40
10				00
11	HERE:	JMP	HERE	C3
12				11
13				00

Ricerca del Primo Carattere Non-Blank

Scopo: Ricerca di un carattere non blank di una stringa di caratteri ASCII (7 bit con il bit più significativo uguale a 0). La stringa inizia nella locazione di memoria 42. Posiziona l'indirizzo del primo carattere non blank nelle locazioni di memoria 40 e 41 (bit più significativi su 41). Un carattere blank è rappresentato dall'esadecimale 20 nel codice ASCII.

Problemi Campione:

```
a.          (42)  =  37 (ASCII 7)
       Risultato  =  (40) = 42
                     (41) = 00
                     poichè la locazione di memoria 42
                     contiene un carattere non blank
b.          (42)  =  20 SP
            (43)  =  20 SP
            (44)  =  20 SP
            (45)  =  46 F
       Risultato  =  (40) = 45
                     (41) = 00
```

Diagramma di Flusso:

Programma Sorgente:

```
        LXI    H,42H    ;PUNTA ALL'INIZIO DELLA STRINGA
        MVI    A,20H    ;PRENDI 'SP' PER CONFRONTO
CHBLK:  CMP    M        ;IL CARATTERE È UN BLANCK?
        JNZ    DONE     ;NO, SALTA
        INX    H
        JMP    CHBLK    ;SÌ, ESAMINA IL CARATTERE SUCCESSIVO
DONE:   SHLD   40H      ;CONSERVA L'INDIRIZZO DEL PRIMO CARATTERE NON BLANK
HERE:   JMP    HERE
```

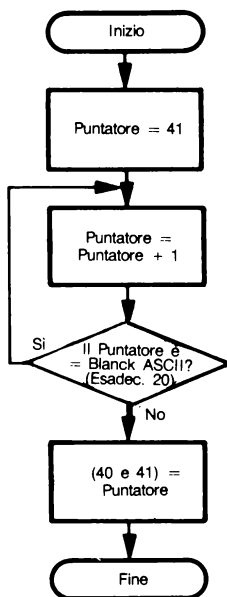
Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		LXI	H,42H	21
01				42
02				00
03		MVI	A,20H	3E
04				20
05	CHBLK:	CMP	M	BE
06		JNZ	DONE	C2
07				0D
08				00
09		INX	H	23
0A		JMP	CHBLK	C3
0B				05
0C				00
0D	DONE:	SHLD	40H	22
0E				40
0F				00
10	HERE:	JMP	HERE	C3
11				10
12				00

Il rilevamento degli spazi di una stringa è un problema usuale. Gli spazi spesso servono per separare i campi oppure frequentemente vengono eliminati dalle stringhe quando vengono usati semplicemente per aumentare la leggibilità o per adattare un particolare formato. È ovviamente prodigo immagazzinare e trasmettere l'inizio, la fine o spazi extra, particolarmente se si sta pagando per le comunicazioni in capability e richieste di memoria.

L'istruzione SHLD è conveniente per l'immagazzinamento di indirizzi nel formato 8080 (prima i bit meno significativi). SHLD 40H immagazzina i contenuti del registro L nella locazione di memoria 40 ed i contenuti del registro H nella locazione 41.

Inoltre se si alterano le condizioni iniziali in modo che la parte di conteggio del ciclo preceda quella di elaborazione, è possibile ridurre il numero di byte del programma ed il tempo di esecuzione del ciclo. Il diagramma di flusso modificato è il seguente:

Diagramma di Flusso:

Programma Sorgente:

```
        LXI     H,41H   ;PUNTA AL BYTE PRIMA DELLA STRINGA
        MVI     A,20H   ;PRENDI 'SP' PER CONFRONTO
NXTCH:  INX     H
        CMP     M       ;IL CARATTERE È UN BLANK?
        JZ      NXTCH   ;SÌ, ESAMINA IL CARATTERE SUCCESSIVO
        SHLD    40H     ;NO, CONSERVA L'INDIRIZZO DEL PRIMO
                        ;CARATTERE NON BLANK
HERE:   JMP     HERE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		LXI	H,41H	21
01				41
02				00
03		MVI	A,20H	3E
04				20
05	NXTCH:	INX	H	23
06		CMP	M	BE
07		JZ	NXTCH	CD
08				05
09				00
0A		SHLD	40H	22
0B				40
0C				00
0D	HERE:	JMP	HERE	C3
0E				0E
0F				00

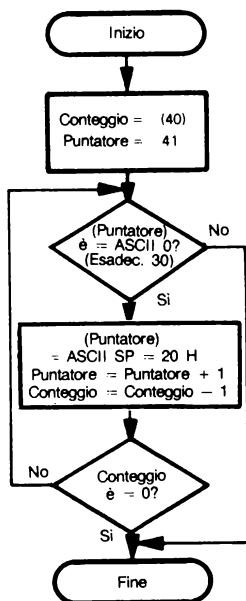
Sostituzione degli Zeri Non Significativi con Blank

Scopo: Redige una stringa di caratteri decimali ASCII in modo da sostituire gli zeri iniziali con blank. La stringa inizia dalla locazione di memoria 41; si assume che essa consista unicamente di cifre decimali codificate ASCII. La lunghezza della stringa è espressa nella locazione di memoria 40.

Problemi Campione:

a. (40) = 02
(41) = 36 = ASCII 6
Risultato = Non cambia poichè la cifra iniziale non è zero

b. (40) = 08
(41) = 30 = ASCII 0
(42) = 30 = ASCII 0
(43) = 38 = ASCII 8
Risultato = (41) = 20 = ASCII SP
(42) = 20 = ASCII SP

Diagramma di Flusso:

Programma Sorgente:

```
        LXI     H,40H
        MOV     B,M     ;CONTEGGIO = LUNGHEZZA STRINGA
        MVI     A,'0'   ;PRENDI ASCII 0 PER CONFRONTO
CHKZ    INX     H
        CMP     M       ;IL DIGIT INIZIALE È 0?
        JNZ     DONE    ;NO, SALTA
        MVI     M,20H   ;SOSTITUISCI LO ZERO INIZIALE CON UN BLANK
        DCR     B       ;SONO STATI ESAMINATI TUTTI I DIGIT?
        JNZ     CHKZ    ;NO, ESAMINA IL DIGIT SUCCESSIVO
DONE:   JMP     DONE
```

La virgoletta singola attorno ai caratteri indica ASCII.

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		LXI	H,40H	21
01				40
02				00
03		MOV	B,M	46
04		MVI	A,'0'	3E
05				30
06	CHKZ:	INX	H	23
07		CMP	M	BE
08		JNZ	DONE	C2
09				11
0A				00
0B		MVI	M,20H	36
0C				20
0D		DCR	B	05
0E		JNZ	CHKZ	C2
0F				06
10				00
11	DONE:	JMP	DONE	C3
12				11
13				00

Si dovranno redigere spesso stringhe decimali prima che vengano stampate o mostrate per migliorare il loro aspetto. I comuni problemi di editing comprendono l'eliminazione di zeri non significativi, posizionamento di numeri, aggiunta di segno od altri caratteri identificanti ed arrotondamento. Chiaramente la stampa di numeri come 0006 o $ 27,34382 può essere fastidiosa e generare confusione.

Qui il ciclo ha due uscite — una si ha se il processore trova una cifra non zero e l'altra si ha se è stata esaminata l'intera stringa.

L'istruzione MVI M,20H posiziona 20H nella locazione di memoria indirizzata dai Registri H ed L. Si potrebbe anche inizializzare il registro C a 20 H (cioè MVI C,20H) ed usare MOV M,C per sostituire lo zero non significativo con un carattere bianco. Si noti il compromesso che si ha in questo esempio. MOV M,C viene eseguita più velocemente di MVI M,20H e così potrebbe diminuire il tempo di esecuzione del ciclo interno. D'altra parte l'istruzione MVI M, 20H è più veloce nella parte di inizializzazione della routine. Se questo esempio fosse impiegato nell'applicazione del registro di cassa quale sequenza converrebbe scegliere e perche?

Tutte le cifre della stringa vengono considerati in codice ASCII cioè le cifre esadecimali vanno da 30 a 39 piuttosto che da 0 a 9 come i decimali ordinari. Quindi per la conversione tra decimale ed ASCII è sufficiente la somma dell'esadecimale 30 con la cifra decimale.

Si può richiedere l'accortezza di usare uno zero nell'eventualità che tutte le cifre siano uguali a zero.

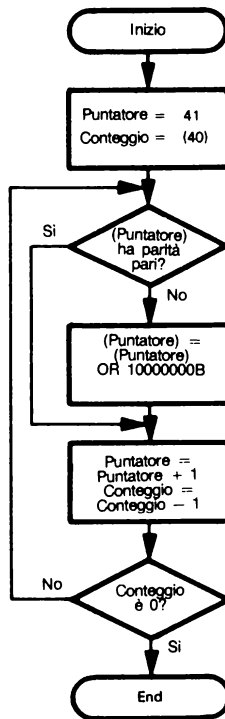
Si noti che ogni cifra ASCII richiede otto bit rispetto ai quattro del sistema BCD. Perciò il codice ASCII è un formato dispendioso per immagazzinare o trasmettere dati numerici.

Aggiunta della Parità Pari ai Caratteri ASCII

Scopo: Aggiungi la Parità Pari ad una stringa di caratteri ASCII a 7 bit. La lunghezza della stringa è contenuta nella locazione di memoria 40 e la stessa stringa inizia dalla locazione di memoria 41. Posiziona la parità pari nel bit più significativo di ogni carattere e lo pone uguale ad 1 se il numero totale di bit 1 nella parola è un numero pari.

Problema Campione:

(40) = 06
(41) = 31
(42) = 32
(43) = 33
(44) = 34
(45) = 35
(46) = 36
Risultato = (41) = B1
(42) = B3
(43) = 33
(44) = B4
(45) = 35
(46) = 36

Diagramma di Flusso:

Programma Sorgente:

```
        LXI     H,40H
        MOV     B,M           ;PRELEVA LA LUNGHEZZA DELLA STRINGA
        MVI     C,10000000B   ;PRELEVA IL BIT PARITÀ DI 1
SETPR:  INX     H
        MOV     A,M           ;PRELEVA UN CARATTERE
        ORA     C             ;POSIZIONA IL BIT PARITÀ AD 1
        JPO     CHCNT         ;PARITÀ È PARI ADESSO?
        MOV     M,A           ;SÌ, CONSERVA IL CARATTERE CON PARITÀ PARI
CHCNT:  DCR     B
        JNZ     SETPR
HERE:   JMP     HERE
```

Programma Oggetto:

Contenuti di Mem. (Esadec.)	Istruzione (Mnemonico)			Indirizzo di Mem. (Esadec.)
00		LXI	H,40H	21
01				40
02				00
03		MOV	B,M	46
04		MVI	C,10000000B	0E
05				80
06	SETPR:	INX	H	23
07		MOV	A,M	7E
08		ORA	C	B1
09		JPO	CHCNT	E2
0A				0D
0B				00
0C		MOV	M,A	77
0D	CHCNT:	DCR	B	05
0E		JNZ	SETPR	C2
0F				06
10				00
11	HERE:	JMP	HERE	C3
12				11
13				00

La Parità è spesso aggiunta ai caratteri ASCII prima della trasmissione su linee rumorose fornendo la possibilità di un semplice controllo di errore. La Parità rivelerà tutti gli errori di singolo bit ma non permette la loro correzione cioè è noto che c'è un errore se la parità ricevuta non corrisponde ma non si può dire quale bit è cambiato.

MVI C,10000000B conserva il bit Parità uguale ad 1 nel registro C. (Si noti l'uso della maschera binaria: lo scopo di detta maschera è più chiaro quando è specificata in questo modo piuttosto che come 80H o 128 decimale).

L'istruzione ORA C pone ad 1 il bit di parità (il più significativo) mentre conserva tutti gli altri bit al valore precedente.

La procedura seguente è usata per determinare se la parità del byte di memoria è pari o dispari. Si opera l'OR del bit di parità con il byte caricato dalla memoria e poi si va a vedere se la parità è dispari. Se la parità è dispari allora il byte della memoria ha parità pari e si procede ai byte rimanenti. Se invece la parità è pari allora il byte di memoria ha parità dispari e perciò si immagazzina nell'Accumulatore il byte di quella locazione di memoria.

I salti condizionati JPO (Salta se Parità dispari) e JPE (salta se Parità Pari) sono impiegati raramente eccetto che nella generazione e controllo parità.

Non si confonda il bit di Parità compreso in ogni carattere ed il bit Parità dell'8080 che è posto a 1 se l'ultimo risultato aritmetico o Booleano ha parità pari.

Confronto di Stringhe di Caratteri

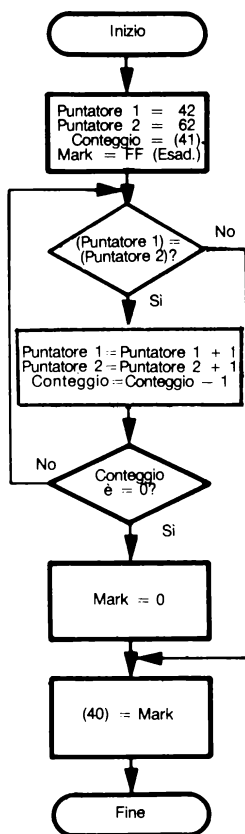
Scopo: Confronta due stringhe di caratteri ASCII per vedere se sono uguali. La lunghezza delle stringhe è espressa dalla locazione di memoria 41, una stringa inizia dalla locazione di memoria 42 e l'altra nella locazione di memoria 62. Se le due stringhe sono uguali la locazione di memoria 40 è azzerata; diversamente viene posta ad FF esadecimale (tutti uni).

Problemi Campione:

a.
(41) = 03
(42) = 43 C
(43) = 41 A
(44) = 54 T
(62) = 43 C
(63) = 41 A
(63) = 54 T
Risultato = (40) = 0 poichè le due stringhe sono uguali

b.
(41) = 03
(42) = 52 R
(43) = 41 A
(44) = 54 T
(62) = 43 C
(63) = 41 A
(64) = 54 T
Risultato = (40) = FF (esadecimale) poichè il primo carattere delle due stringhe è diverso

Si noti che il processo di confronto termina non appena la CPU trova una differenza – il resto della stringa non viene esaminato.

Diagramma di Flusso

Programma Sorgente:

```
        LXI   H,A41H
        MOV   B,M     ;CONTEGGIO = LUNGHEZZA DELLE STRINGHE
        INX   H       ;PUNTATORE 1 = INIZIO STRINGA 1
        LXI   D,62H   ;PUNTATORE 2 = INIZIO STRINGA 2
        MVI   C,OFFH  ;MARK = FF (ESADECIMALE)
CHCAR:  LDAX  D       ;PRELEVA UN CARATTERE DALLA STRINGA 2
        CMP   M       ;SONO UGUALI?
        JNZ   DONE    ;NO, DONE
        INX   D
        INX   H
        DCR   B       ;SONO STATI CONTROLLATI TUTTI I CARATTERI?
        JNZ   CHCAR   ;NO, CONTROLLA LA COPPIA SUCCESSIVA
        MVI   C,0     ;SÌ, CONFRONTO COMPLETO, MARK = 0
DONE:   MOV   A,C
        STA   40H     ;CONSERVA MARK
HERE:   JMP   HERE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		LXI	H,41H	21
01				41
02				00
03		MOV	B,M	46
04		INX	H	23
05		LXI	D,62H	11
06				62
07				00
08		MVI	C,0FFH	0E
09				FF
0A	CHCAR:	LDAX	D	1A
0B		CMP	M	BE
0C		JNZ	DONE	C2
0D				17
0E				00
0F		INX	D	13
10		INX	H	23
11		DCR	B	05
12		JNZ	CHCAR	C2
13				0A
14				00
15		MVI	C,0	0E
16				00
17	DONE:	MOV	A,C	79
18		STA	40H	32
19				40
1A				00
1B	HERE:	JMP	HERE	C3
1C				1B
1D				00

Il confronto di stringhe di caratteri ASCII è una parte essenziale per l'analisi di comandi, riconoscimento di nomi, identificazione di variabili o codici di operazione in assemblatori e compilatori, ricerca di liste e molti altri compiti.

Il programma impiega due puntatori, uno localizzato nei registri H ed L e l'altro nei registri D ed E. Le uniche istruzioni che impiegano l'indirizzo in D ed E sono LDAX (Carica Indiretto l'Accumulatore) e STAX (Immagazzina Indiretto l'Accumulatore); invece le operazioni aritmetiche e logiche con la memoria ed i trasferimenti dalla memoria ad altri registri (per esempio ADD M, AND M, MOV B,M, ecc.) possono essere seguiti solo impiegando l'indirizzo contenuto nei registri H ed L.

L'ordine delle operazioni é molto importante a causa del piccolo numero di istruzioni che fanno uso degli indirizzi contenuti in D ed E. È necessario muovere un carattere dalla stringa puntata da D ed E all'Accumulatore e paragonarlo con un carattere della stringa puntata da H ed L. Questo artificio è necessario perché l'8080 non é dotato di un'istruzione che consenta un confronto di un carattere contenuto in una stringa puntata da D ed E. Per esempio se si sostituisce LDAX con MOV A,M quale dovrebbe essere la successiva istruzione? Questa asimmetria è peculiare dell'8080 e può originare errori di programmazione.

Si noti che ogni iterazione aggiorna entrambi i puntatori.

Questo programma potrebbe avvantaggiarsi del fatto che un registro è noto essere zero dopo che è stato eseguito un particolare salto condizionato. Quando viene eseguita l'istruzione JNZ CHCAR se non viene eseguito il salto il registro B contiene zero. Perciò si può muovere B in C, il registro di flag, per indicare che è stata trovata un'uguaglianza di caratteri.

PROBLEMI

1) Lunghezza di un Messaggio della Telescrivente

Scopo: Determina la lunghezza di un messaggio ASCII. Tutti i caratteri sono ASCII a 7 bit con quello MBS uguale a 0. La stringa di caratteri nella quale é contenuto il messaggio stesso inizia con un carattere STX ASCII (esadecimale 02) e termina con ETX (esadecimale 03). Posiziona la lunghezza del messaggio (il numero di caratteri tra STX ed ETX, entrambi non compresi) nella locazione di memoria 40.

Problema Campione:

```
    (41)  =  49
    (42)  =  02 STX
    (43)  =  47 G
    (44)  =  4F 0
    (45)  =  03 ETX
Risultato =  (40) = 02 poichè esistono due caratteri tra STX
             della locazione 42 ed ETX della locazione 45
```

2) Ricerca dell'Ultimo Carattere Non-Blank

Scopo: Analizza una stringa ASCII per ricercare l'ultimo carattere non blank. La stringa inizia nella locazione di memoria 42 e termina con un carattere di ritorno carrello (esadecimale 0D). Posiziona l'indirizzo dell'ultimo carattere non blank nelle locazioni di memoria 40 e 41 (bit più significativi in 41).

Problemi Campione:

```
a.      (42)  =  37 (ASCII 7)
        (43)  =  0D 'CR'
    Risultato =  (40) = 42
                 (41) = 00
                 poichè l'ultimo (e l'unico) carattere non blank
                 si trova nella locazione di memoria 42
b.      (42)  =  41 A
        (43)  =  20 'SP'
        (44)  =  48 H
        (45)  =  41 A
        (46)  =  54 T
        (47)  =  20 'SP'
        (48)  =  20 'SP'
        (49)  =  0D 'CR'
    Risultato =  (40) = 46
                 (41) = 00
```

3) Troncamento di una Stringa Decimale alla Forma Intera

Scopo: Redige una stringa di caratteri decimali ASCII sostituendo tutte le cifre alla destra del punto decimale con caratteri blank ASCII (esadecimale 20). La stringa inizia nella locazione di memoria 41 e si assume composta interamente da cifre decimali codificate ASCII ed eventualmente da un punto decimale (esadecimale 2E). La lunghezza della stringa è indicata nella locazione di memoria 40. Se nella stringa non compare il punto decimale si assume che tutte le cifre sono numeri interi con il punto decimale (implicito) all'estrema destra.

Problemi Campione:

a.
(40) = 04
(41) = 37 7
(42) = 2E .
(43) = 38 8
(44) = 31 1
Risultato = (41) = 37 7
(42) = 2E .
(43) = 20 'SP'
(44) = 20 'SP'

b.
(40) = 03
(41) = 36 6
(42) = 37 7
(43) = 31 1
Risultato = Invariato perchè il numero è assunto essere 671

4) Controllo di Parità Pari di Caratteri ASCII

Scopo: Controlla la Parità Pari di una stringa di caratteri ASCII. La lunghezza della stringa è indicata nella locazione di memoria 41 e la stessa stringa inizia dalla locazione di memoria 42. Se la parità di tutti i caratteri della stringa è corretta, azzera la locazione di memoria 40, diversamente pone i contenuti della locazione di memoria 40 ad FF esadecimale (tutti uni).

Problemi Campione:

a.
(41) = 03
(42) = B1
(43) = B2
(44) = 33
Risultato = (40) = 00 poichè tutti i caratteri hanno parità pari

b.
(41) = 03
(42) = B1
(43) = B6
(44) = 33
Risultato = (40) = FF (esadecimale) poichè il carattere della locazione 42 non ha parità pari

5) Confronto di Stringhe

Scopo: Confronta due stringhe di caratteri ASCII per vedere qual è la maggiore (cioè quale segue l'altra in ordine 'alfabetico'). La lunghezza delle stringhe è indicata nella locazione di memoria 41; una stringa inizia dalla locazione 42 e l'altra dalla 62. Se la stringa iniziante nella locazione 42 è maggiore o uguale all'altra azzera la locazione di memoria 40; diversamente pone la locazione 40 ad FF esadecimale (tutti uni).

Problemi Campione:

a.	(41)	=	03
	(42)	=	43 C
	(43)	=	41 A
	(44)	=	54 T
	(62)	=	42 B
	(63)	=	41 A
	(64)	=	54 T
	Risultato	=	(40) = 00 poichè CAT è 'maggiore' di BAT
b.	(41)	=	03
	(42)	=	44 D
	(43)	=	4F 0
	(44)	=	47 G
	(62)	=	44 D
	(63)	=	4F 0
	(64)	=	47 G
	Risultato	=	(40) = 00 poichè le due stringhe sono uguali
c.	(41)	=	03
	(42)	=	43 C
	(43)	=	41 A
	(44)	=	54 T
	(62)	=	43 C
	(63)	=	55 U
	(64)	=	54 T
	Risultato	=	(40) = FF (esadecimale) poichè CUT è 'maggiore' di CAT

Capitolo 7
CONVERSIONE DI CODICE

La conversione di codice è un problema continuo per i microprocessori. I dispositivi periferici forniscono dati in codici ASCII, BCD o speciali. Il computer deve essere predisposto per la conversione di questi dati in binario o decimale in modo da poterli elaborare. I dispositivi di Uscita richiedono i dati in ASCII, BCD, 7-segmenti od altri codici. Conseguentemente il computer deve convertire questi risultati in una forma opportuna dopo che è stata completata l'elaborazione. Alcune conversioni di codice sono semplici da eseguire in hardware; per esempio esistono circuiti integrati standard per la conversione da BCD a 7-segmenti. I Ricevitori/Trasmettitori Asincroni Universali (UART) operano la conversione tra i dati ASCII ed i formati delle telescriventi. Comunque al programma vengono ancora richiesti molti lavori di conversione.

ESEMPI

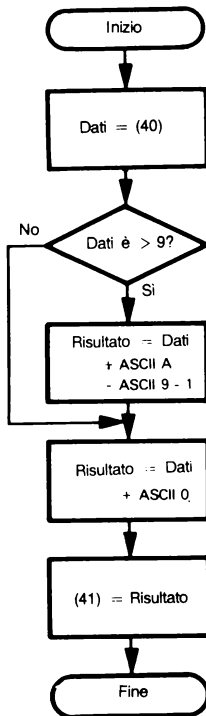
Da Esadecimale ad ASCII

Scopo: Converte i contenuti della locazione di memoria 40 in un carattere ASCII. La locazione di memoria 40 contiene una singola cifra esadecimale (i quattro bit più significativi sono zero). Immagazzina il carattere ASCII nella locazione di memoria 41.

Problemi Campione:

a. (40) = 0C
Risultato = (41) = 43 'C'

b. (40) = 06
Risultato = (41) = 36 '6'

Diagramma di Flusso:

Programma Sorgente:

```
        LDA     40H         ;ACCETTA DATI
        CPI     10          ;I DATI SONO 10 O PIÙ?
        JNC     ASCZ
        ADI     'A'-'9'-1   ;SÌ, AGGIUNGI OFFSET PER LE LETTERE
ASCZ:   ADI     '0'         ;AGGIUNGI OFFSET PER ASCII
        STA     41H         ;IMMAGAZZINA IL RISULTATO ASCII
HERE:   JMP     HERE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)	Contenuti di Mem. (Esadec.)
00	LDA 40H	3A
01		40
02		00
03	CPI 10	FE
04		0A
05	JNC ASCZ	D2
06		0A
07		00
08	ADI 'A'-'9'-1	C6
09		07
0A	ASCZ: ADI '0'	C6
0B		30
0C	STA 41H	32
0D		41
0E		00
0F	HERE: JMP HERE	C3
10		0F
11		00

In questo programma l'idea fondamentale è quella di sommare l'ASCII 0 a tutte le cifre esadecimali. Questa somma converte correttamente le cifre decimali; comunque c'è un vuoto tra ASCII 9 (39 esadecimale) ed ASCII A (41 esadecimale). Questo vuoto deve essere sommato di digit non decimali, cioè A, B, C, D, E, F. Questo problema viene risolto dall'istruzione ADI che somma l'offset 'A'-'9'-1 ai contenuti: dell'Accumulatore. Si provi a giustificare perchè l'offset è 'A'-'9'-1.

Si noti che i termini dell'addizione sono impostati nel programma in linguaggio assembler sotto forma ASCII (la virgoletta singola i caratteri ASCII). Gli offset per le lettere sono lasciati indicati come un'espressione aritmetica. Questo sforzo mira ad avere dei termini più chiari possibile nella lista del linguaggio assembly. L'ulteriore tempo di assembly è un prezzo molto piccolo pagato per un grosso aumento in chiarezza.

Questa routine potrebbe essere impiegata in molti programmi; per esempio i programmi monitor devono convertire i digit esadecimali in ASCII allo scopo di mostrare i contenuti delle locazioni di memoria, espressi in esadecimale, ad una stampante ASCII oppure ad un display CRT.

Un altro metodo di conversione (più veloce) che richiede solo salti incondizionati è mostrato nel seguente programma descritto da Allison nella rivista COMPUTER (vedere ALLISON D.R. «A Design Philosophy for Microcomputer Architectures», Computer, Febbraio 1977, pp 35-41. Questo è eccellente articolo di cui si raccomanda la lettura).

```
        LDA   40H    ;ACCETTA IL DIGIT ESADECIMALE
        ADI   90H    ;SOMMA DECIMALE 90 BCD
        DAA
        ACI   40H    ;SOMMA DECIMALE 40 BCD + CARRY
        DAA
        STA   41H    ;IMMAGAZZINA IL DIGIT ASCII
HERE:   JMP   HERE
```

Si provi questo programma su qualche digit. Si provi a giustificare come lavora.

Da Decimale a 7-Segmenti

Scopo: Converte i contenuti della locazione di memoria 40 ad un codice a 7-segmenti nella locazione di memoria 41. Se la locazione di memoria 40 non contiene una sola cifra decimale azzera la locazione 41.

Si può usare la seguente tabella per convertire i numeri decimali in codice a 7-segmenti. Il codice a 7-segmenti è organizzato con il bit più significativo sempre a zero seguito dal codice (1 = on, 0 = off) per i segmenti g, f, e, d, c, b, a (Vedere Figura 7-1).

Cifra	Codice
0	3F
1	06
2	5B
3	4F
4	66
5	6D
6	7D
7	07
8	7F
9	6F

Si noti che la tabella impiega 7D per 6 piuttosto che l'alternativo 7C (segmento in alto off) per evitare confusione con la lettera minuscola b e 6F per 9 piuttosto che 67 (segmento in basso off) per nessuna ragione particolare.

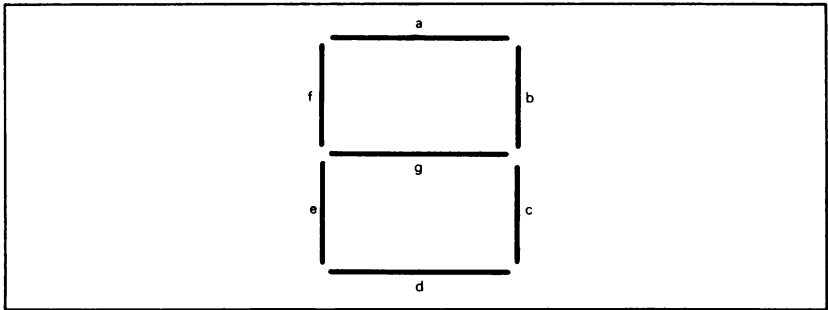
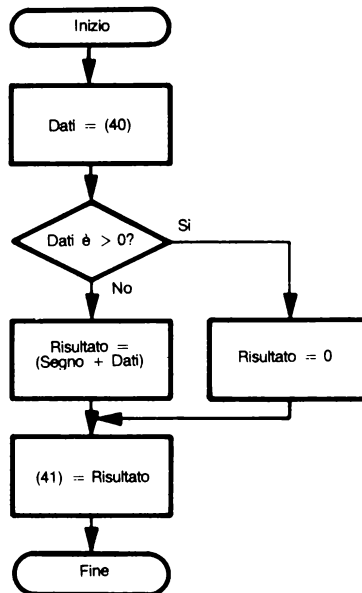



Figura 7-1
Disposizione Dei 7 Segmenti

Problemi Campione:

a. (40) = 03
Risultato = (41) = 4F

b. (40) = 28
Risultato = (41) = 00

Diagramma di Flusso:

Programma Sorgente:

```
        MVI    B,0      ;ACCETTA CODICE ERRORE COME BLANK DISPLAY
        LDA    40H      ;ACCETTA DATI
        CPI    10       ;I DATI SONO >9?
        JNC    DONE     ;SÌ, DONE
        LXI    H,SSEG   ;INDIRIZZO BASE DELLA TABELLA A 7-SEGMENTI
        MOV    C,A      ;ESEGUE DATI IN UN INDICE A 16-BIT
        DAD    B        ;TROVA L'ELEMENTO INDIRIZZATO
DONE:   MOV    B,M      ;ACCETTA IL CODICE A 7-SEGMENTI
        MOV    A,B
        STA    41H      ;IMMAGAZZINA IL CODICE A 7-SEGMENTI O
                        ;ZERO SE ERRORE
HERE:   JMP    HERE
SSEG:   DB     3FH,06H,5BH,4FH,66H
        DB     6DH,7DH,07H,7FH,6FH
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		MVI	B,0	06
01				00
02		LDA	40H	3A
03				40
04				00
05		CPI	10	FE
06				0A
07		JNC	DONE	D2
08				10
09				00
0A		LXI	H,SSEG	21
0B				17
0C				00
0D		MOV	C,A	4F
0E		DAD	B	09
0F		MOV	B,M	46
10	DONE:	MOV	A,B	78
11		STA	41H	32
12				41
13				00
14	HERE:	JMP	HERE	C3
15				14
16				00
17	SSEG	DB	3FH,06H,5BH,4FH,66H	3F
18				06
19				5B
1A				4F
1B				66
1C		DB	6DH,7DH,07H,7FH,6FH	6D
1D				7D
1E				07
1F				7F
20				6F

Il programma calcola l'indirizzo di memoria del codice desiderato sommando l'indice (cioè il digit che deve essere posizionato sul display) all'indirizzo base della tabella del codice a sette-segmenti. Questa procedura è nota come tabella di consultazione (table look-up).
La pseudo-operazione in linguaggio assembly DB (Definisci Byte) viene impiegata per posizionare i dati costanti nella memoria di programma. Tali dati possono comprendere tabelle, titoli, messaggi di errore, messaggi di innesco, caratteri di format, soglie, ecc.. La label congiunta con una pseudo-operazione DB è assegnata al valore dell'indirizzo nel quale l'assemblatore posiziona il primo byte di dati. L'assemblatore posiziona semplicemente i dati della tabella nella memoria. Una pseudo-operazione DB si risolve nel riempimento di una o più locazioni di memoria.
Le tabelle sono spesso impiegate per eseguire conversioni di codice quando la relazione funzionale non è così semplice come quella dell'esempio di conversione da Esadecimale ad ASCII. Tale tabella normalmente contiene tutti i risultati organizzati secondo i dati d'ingresso cioè il primo ingresso ha il codice corrispondente al numero zero.
Il codice a 7-segmenti è normalmente impiegato nei digit dei display assieme a poche lettere ed altri caratteri. I display a 7-segmenti del tipo dei computer sono non dispendiosi, facili da combinare ed impiegano poca potenza; comunque i digit del codice a 7-segmenti possono essere difficili da leggere.

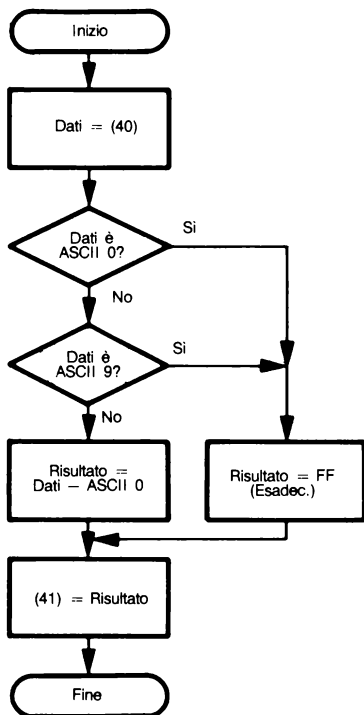
Da ASCII a Decimale

Scopo: Converte i contenuti della locazione di memoria 40 da un carattere ASCII ad una cifra decimale ed immagazzina il risultato nella locazione di memoria 41. Se i contenuti della locazione 40 non corrispondono al Carattere ASCII di una cifra decimale pone i contenuti della locazione di memoria 41 ad FF (esadec.).

Problemi Campione:

a. (40) = 37 (ASCII 7)
Risultato = (41) = 07

b. (40) = 55
Risultato = (41) = FF

Diagramma di Flusso:

Programma Sorgente:

```
        MVI     B,0FFH  ;ACCETTA IL SEGNALATORE DI ERRORE
                        ;(FF ESADEC.)
        LDA     40H     ;ACCETTA I DATI
        SUI     '0'     ;SOTTRAI ASCII 0
        JC      DONE    ;NO DIGIT SE < ASCII 0
        CPI     10      ;IL RISULTATO È ≥ 0?
        JNC     DONE    ;NO DIGIT SE ≥ 10
        MOV     B,A
DONE:   MOV     A,B
        STA     41H     ;IMMAGAZZINA IL RISULTATO DECIMALE
                        ;O SEGNALATORE D'ERRORE
HERE:   JMP     HERE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		MVI	B,0FFH	06
01				FF
02		LDA	40H	3A
03				40
04				00
05		SUI	'0'	D6
06				30
07		JC	DONE	DA
08				10
09				00
0A		CPI	10	FE
0B				10
0C		JNC	DONE	D2
0D				10
0E				00
0F		MOV	B,A	47
10	DONE:	MOV	A,B	78
11		STA	41H	32
12				41
13				00
14	HERE:	JMP	HERE	C3
15				14
16				00

Questo programma determina se i dati sono compresi tra ASCII 0 ed ASCII 9 compresi. Se è così i dati sono la rappresentazione ASCII di un decimale poichè le cifre formano una sequenza. La sottrazione dell'esadecimale 30 origina il decimale equivalente.

Si noti che viene eseguito un confronto con una sottrazione effettiva (SUI '0') poichè questa sottrazione è necessaria per convertire l'ASCII in decimale. Il secondo confronto viene eseguito con una sottrazione implicata (CPI 10) poichè il risultato decimale è ora nell'Accumulatore e non si vuole cambiarlo.

Questo tipo di programma, è pratico in molte situazioni. Per esempio la conversione da ASCII a decimale è necessaria quando devono entrare dei numeri decimali da un dispositivo ASCII come una telescrivente od un terminale CRT.

Da BCD a Binario

Scopo: Converte i due digit BCD delle locazioni di memoria 40 e 41 in un numero binario posizionato nella locazione 42. Il digit più significativo BCD è quello della locazione 40.

Problemi Campione:

a. (40) = 02
(41) = 09
Risultato = (42) = 1D (esadec.) = 29 (decimale)

b. (40) = 07
(41) = 01
Risultato = (42) = 47 (esadec.) = 71 (decimale)

Programma Sorgente:

```
        LXI     H,40H
        MOV     A,M     ;ACCETTA IL DIGIT PIÙ SIGNIFICATIVO (MSD)
        ADD     A       ;MSD X 2
        MOV     B,A     ;CONSERVA MSD X 2
        ADD     A       ;MSD X 4
        ADD     A       ;MSD X 8
        ADD     B       ;MSD X 10
        INX     H       ;PUNTA AL DIGIT MENO SIGNIFICATIVO
        ADD     M       ;SOMMA ALLA FORMA BINARIA EQUIVALENTE
        INX     H
        MOV     M,A     ;IMMAGAZZINA IL BINARIO EQUIVALENTE
HERE:   JMP     HERE
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)	Contenuti di Memoria (Esadec.)
00	LXI H,40H	21
01		40
02		00
03	MOV A,M	7E
04	ADD A	87
05	MOV B,A	47
06	ADD A	87
07	ADD A	87
08	ADD B	80
09	INX H	23
0A	ADD M	86
0B	INX H	23
0C	MOV M,A	77
0D	HERE: JMP HERE	C3
0E		0D
0F		00

Questo programma moltiplica il digit BCD della locazione 40 per dieci impiegando addizioni ripetute. Si noti che ADD A moltiplica per 2 il contenuto dell'Accumulatore. Questo permette la moltiplicazione dell'Accumulatore con piccoli numeri decimali in poche istruzioni. Come si potrebbe moltiplicare l'Accumulatore per 16? per 12? per 7?

Gli ingressi BCD vengono convertiti in binario allo scopo di immagazzinamento e calcoli. I numeri decimali richiedono spazio addizionale di memoria e calcoli più complessi. Comunque, la conversione può eliminare alcuni dei vantaggi dell'aritmetica binaria.

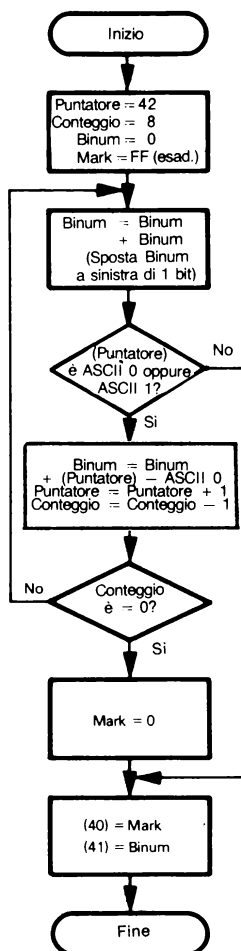
I numeri BCD richiedono circa il 20% di memoria in più dei numeri binari, per esempio 1000 richiede 3 digit BCD (12 bit) e 10 bit in binario (poichè $2^{10} = 1024 \approx 1000$).

Da Stringhe ASCII a Numeri Binari

Scopo: Converte una stringa di 8 caratteri ASCII in un numero binario ed immagazzina il risultato nella locazione di memoria 41. Se ciascuno dei caratteri non è ASCII zero nè ASCII uno, pone la locazione di memoria 40 ad FF (esadec.); diversamente azzera la locazione di memoria 40. La stringa di caratteri è nelle locazioni da 42 a 49 con il bit più significativo nella locazione di memoria 42.

Problemi Campione:

a.	(42)	=	30	('0')
	(43)	=	31	('1')
	(44)	=	31	('1')
	(45)	=	30	('0')
	(46)	=	30	('0')
	(47)	=	30	('0')
	(48)	=	30	('0')
	(49)	=	31	('1')
	Risultato	=	(40) = 0	
			(41) = 61 (cioè 01100001)	
b.	Lo stesso di a. eccetto:			
	(45)	=	34	('4')
	Risultato	=	(40) = FF	

Diagramma di Flusso:

Programma Sorgente:

```
        LXI    H,42H    ;PUNTA ALL'INIZIO DELLA STRINGA
        SUB    A        ;IL NUMERO BINARIO = 0
        MVI    B,8      ;CONTEGGIO = 8
        MVI    C,0FFH   ;MARKER = INDICATORE D'ERRORE
BICON:  ADD    A        ;SPOSTA A SINISTRA IL NUMERO BINARIO
        MOV    D,A
        MOV    A,M      ;ACCETTA IL CARATTERE DALLA STRINGA
        SUI    '0'      ;CONVERTE A BINARIO
        CPI    2        ;IL RISULTATO È MINORE DI 2?
        JNC    DONE     ;NO, ERRORE NEL NUMERO BINARIO
        ADD    D        ;SOMMA BIT AL NUMERO BINARIO
        INX    H
        DCR    B        ;TUTTI I CARATTERI SONO CONVERTITI?
        JNZ    BICON
        MVI    C,0      ;SÌ, MARKER = 0, NUMERO BINARIO CORRETTO
DONE:   LXI    H,40H
        MOV    A,M
        INX    H
        MOV    M,C      ;MARK SE CORRETTO OPPURE NO
HERE:   JMP    HERE
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)			Contenuti di Memoria (Esadec.)
00		LXI	H,42H	21
01				42
02				00
03		SUB	A	97
04		MVI	B,8	06
05				08
06		MVI	C,0FFH	0E
07				FF
08	BICON:	ADD	A	87
09		MOV	D,A	57
0A		MOV	A,M	7E
0B		SUI	'0'	D6
0C				30
0D		CPI	2	FE
0E				02
0F		JNC	DONE	D2
10				1A
11				00
12		ADD	D	82
13		INX	H	23
14		DCR	B	05
15		JNZ	BICON	C2
16				08
17				00
18		MVI	C,0	0E
19				00
1A	DONE:	LXI	H,40H	21
1B				40
1C				00
1D		MOV	A,M	77
1E		INX	H	23
1F		MOV	M,C	71
20	HERE:	JMP	HERE	C3
21				20
22				00

L'istruzione ADD A è uno spostamento logico a sinistra, cioè, essa muove i bit precedentemente determinati a sinistra di una posizione ed azzera il bit meno significativo.

Poiche (B) = 0 quando tutti i digit ASCII sono stati convertiti si potrebbe sostituire MVI C,0 con MVI C,B (perchè?). Il guadagno è un byte di codice ed un piccolo tempo di esecuzione. Un'altra alternativa potrebbe essere INR C (perché?).

La conversione dall'ASCII comprende semplicemente la sottrazione dell'ASCII 0 (esadec. 30).

L'istruzione CPI 2 pone:

Carry = 1 se (A) $<$ 2 cioè (A) = 0 oppure 1
Carry = 0 se (A) $\geq$ 2

L'istruzione ADD D somma il resto del numero che si sta costituendo nell'Accumulatore.

Poichè l'Accumulatore contiene uno zero oppure un uno e D contiene uno zero nel bit meno significativo, questa istruzione aggiorna il numero che si sta costruendo.

La conversione dall'ASCII al binario è richiesta naturalmente quando i numeri di ingresso sono in forma binaria da un dispositivo ASCII come un assemblatore che quindi genera ingressi binari.

PROBLEMI

1) Da ASCII ad Esadecimale

Scopo: Converte i contenuti della locazione di memoria 40 in un digit esadecimale ed immagazzina il risultato nella locazione 41. Si assume che la locazione 40 contenga la rappresentazione ASCII di un digit esadecimale (7 bit con MSB uguale a 0).

Problemi Campione:

a.	(40)	=	43 'C'
	Risultato	=	(41) = 0C
b.	(40)	=	36 '6'
	Risultato	=	(41) = 06

2) Da 7-Segmenti a Decimale

Scopo: Converte i contenuti della locazione di memoria 40 da un codice a 7-segmenti ad un numero decimale posizionato nella locazione 41. Se la locazione 40 non contiene un codice a 7-segmenti valido pone la locazione 41 ad FF (Esadec.). Impiega la tabella per i 7-segmenti fornita nell'esempio da Decimale a 7-segmenti e fa la conversione per confronto.

Problemi Campione:

a.	(40)	=	4F
	Risultato	=	(41) = 03
b.	(40)	=	28
	Risultato	=	(41) = FF

3) Da Decimale ad ASCII

Scopo: Converte i contenuti della locazione di memoria 40 da una cifra decimale ad un carattere ASCII ed immagazzina il risultato nella locazione di memoria 41. Se il numero nella locazione 40 non è una cifra decimale pone i contenuti della locazione 41 uguale ad un carattere blank ASCII (20 esadec.).

Problemi Campione:

a.	(40)	=	07
	Risultato	=	(41) = 37 ('7')
b.	(40)	=	55
	Risultato	=	(41) = 20 ('SP')

4) Da Binario a BCD

Scopo: Converte un numero binario della locazione di memoria 40 in due digit BCD nelle locazioni 41 e 42 (digit più significativo in 41). Il numero della locazione 40 è tacitamente considerato minore di 100.

Problemi Campione:

a.	(40)	=	1 D (29 decimale)
	Risultato	=	(41) = 02
			(42) = 09
b.	(40)	=	47 (71 decimale)
	Risultato	=	(41) = 07
			(42) = 01

5) Da Numero Binario a Stringa ASCII

Scopo: Converte i contenuti della locazione di memoria 41 in una stringa di caratteri ASCII rappresentanti i singoli bit. La stringa occupa le locazioni da 42 a 49 con il bit più significativo nella locazione 42.

Problema Campione:

(41)	=	61 (cioè 01100001)
Risultato	=	(42) = 30 ('0')
		(43) = 31 ('1')
		(44) = 31 ('1')
		(45) = 30 ('0')
		(46) = 30 ('0')
		(47) = 30 ('0')
		(48) = 30 ('0')
		(49) = 31 ('1')

Capitolo 8
PROBLEMI ARITMETICI

La maggior parte delle applicazioni aritmetiche del microprocessore consistono nella manipolazione di parole multiple binarie o decimali. Una correzione decimale (Aggiustamento Decimale) e qualche altra permettono operazioni aritmetiche decimali e sono le sole istruzioni aritmetiche oltre alle operazioni base di addizione e sottrazione. È necessario realizzare le altre operazioni aritmetiche in software.

Le operazioni aritmetiche binarie a precisione multipla richiedono una semplice ripetizione delle istruzioni base di singola parola. Il bit Carry trasferisce informazioni tra una parola e l'altra. Somma con Carry e Sottrai con Prestito impiegano l'informazione dell'operazione aritmetica precedente. Occorre assicurarsi di azzerare Carry prima dell'esecuzione sulle prime parole (ovviamente non c'è Carry o Prestito dai bit meno significativi).

L'operazione aritmetica decimale è un compito abbastanza comune per i microprocessori che, in gran parte sono dotati di una speciale istruzione per questo scopo. Questa istruzione può eseguire direttamente un'addizione decimale oppure trasformare il risultato di un'addizione binaria nella forma decimale corretta. L'operazione aritmetica decimale è essenziale in applicazioni come terminali in punti di vendita, calcolatori, processori di controllo, sistemi ad ordine di ingresso e terminali bancari.

È possibile realizzare e la moltiplicazione e la divisione come serie di addizioni e sottrazioni, rispettivamente in modo simile alla procedura manuale. Operazioni in doppia-parola qui diventano importanti perchè una moltiplicazione produce un risultato circa il doppio più lungo degli operandi mentre la divisione contrae la lunghezza del risultato. Le moltiplicazioni e divisioni impiegano un tempo notevole se eseguite in software a causa della ripetizione richiesta di operazioni aritmetiche e di scorrimento.

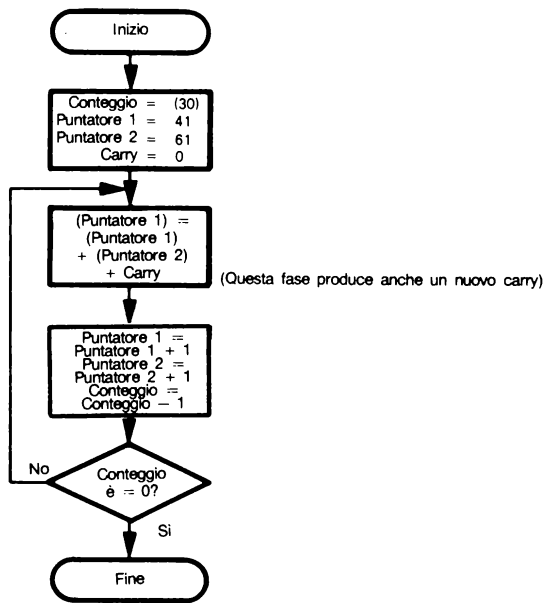
ESEMPI

Addizione a Precisione Multipla

Scopo: Somma due numeri binari a precisione multipla. La lunghezza dei numeri (in byte) è indicata nella locazione di memoria 30, gli stessi numeri iniziano (prima i bit meno significativi) nelle locazioni di memoria 41 e 61 rispettivamente, e la somma sostituisce il numero iniziante nella locazione di memoria 41.

Problema Campione:

```
     (30)  =  04
     (41)  =  C3
     (42)  =  A7
     (43)  =  5B
     (44)  =  2F
     (61)  =  B8
     (62)  =  35
     (63)  =  DF
     (64)  =  14
Risultato  =  (41) = 7B
              (42) = DD
              (43) = 3A
              (44) = 44
    cioè:     2F5BA7C3
           +  14DF35B8
              --------
              443ADD78
```

Diagramma di Flusso:

Programma Sorgente:

```
        LDA     30H     ;CONTEGGIO = LUNGHEZZA DELLE STRINGHE  (IN BYTE)
        MOV     B,A
        LXI     H,41H   ;INIZIA PUNTATORE 1 DALLA PRIMA PAROLA
                        ;DELLA STRINGA 1
        LXI     D,61H   ;INIZIA PUNTATORE 2 DALLA PRIMA PAROLA
                        ;DELLA STRINGA 2
        ANA     A       ;AZZERA INIZIALMENTE CARRY
ADDW:   LDAX    D       ;ACCETTA PAROLA DALLA STRINGA 2
        ADC     M       ;SOMMA LE PAROLE DALLA STRINGA 1
        MOV     M,A     ;IMMAGAZZINA IL RISULTATO
        INX     D
        INX     H
        DCR     B
        JNZ     ADDW
HERE:   JMP     HERE
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)	Contenuti di Memoria (Esadec.)
00	LDA 30H	3A
01		30
02		00
03	MOV B,A	47
04	LXI H,41H	21
05		41
06		00
07	LXI D,61H	11
08		61
09		00
0A	ANA A	A7
0B	ADDW: LDAX D	1A
0C	ADC M	8E
0D	MOV M,A	77
0E	INX D	13
0F	INX H	23
10	DCR B	05
11	JNZ ADDW	C2
12		0B
13		00
14	HERE: JMP HERE	C3
15		14
16		00

L'istruzione ANA A viene impiegata per azzerare il bit Carry. Qualsiasi altra operazione logica potrebbe avere lo stesso effetto. Il Carry deve essere azzerato poichè non può esistere riporto nell'addizione dei byte meno significativi.

L'istruzione ADC, Addizione con Carry, fa uso del Carry proveniente dalle parole precedenti della stessa addizione. ADC è la sola istruzione del ciclo che influenza il Carry. Si ricordi infatti che INX e DCR non influenzeranno il Carry.

Entrambi i puntatori, quello dei registri D ed E e quello di H ed L, devono essere aggiornati durante ogni iterazione.

PRECISIONE DECIMALE IN BINARIO

Questa procedura può sommare numeri binari di qualunque lunghezza. Si noti che dieci bit binari corrispondono a tre cifre decimali essendo $2^{10} = 1024 \approx 1000$. Così si può calcolare il numero di bit richiesti per ottenere una certa precisione in cifre decimali. Per esempio

La precisione di dieci cifre decimali richiede:

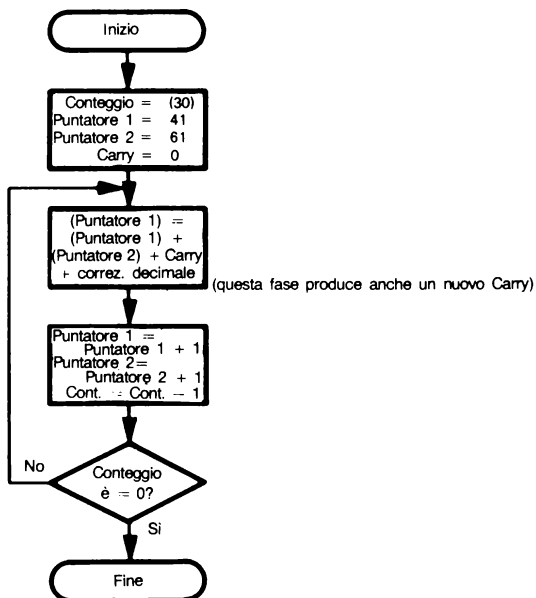
$$10 \times \frac{10}{3} \approx 33 \text{ bit}$$

Addizione Decimale

Scopo: Somma due numeri (BCD) decimali a parola multipla. La lunghezza dei numeri è indicata nella locazione di memoria 30, i numeri stessi iniziano (prima i bit meno significativi) dalle locazioni di memoria 41 e 61 rispettivamente e la somma sostituisce il numero iniziante della locazione 41.

Problema Campione:

```
      (30)  =  04
      (41)  =  85
      (42)  =  19
      (43)  =  70
      (44)  =  36
      (61)  =  59
      (62)  =  34
      (63)  =  66
      (64)  =  12
 Risultato  =  (41)  =  44
               (42)  =  54
               (43)  =  36
               (44)  =  49
     cioè:     36701985
            +  12663459
               49365444
```

Diagramma di Flusso

Programma Sorgente:

```
        LDA     30H     ;CONTEGGIO = LUNGHEZZA DELLE STRINGHE
                        ;(IN BYTE)
        MOV     B,A
        LXI     H,41H   ;PUNTATORE 1 = PRIMA PAROLA DELLA STRINGA 1
        LXI     D,61H   ;PUNTATORE 2 = PRIMA PAROLA DELLA STRINGA 2
        ANA     A       ;AZZERA INIZIALMENTE CARRY
DECAD:  LDAX    D       ;ACCETTA 2 DIGIT DALLA STRINGA 2
        ADC     M       ;SOMMA UNA COPPIA DI DIGIT DALLA STRINGA 1
        DAA             ;ESEGUE DECIMALE ADDIZIONALE
        MOV     M,A     ;IMMAGAZZINA IL RISULTATO COME NUOVA STRINGA 1
        INX     D
        INX     H
        DCR     B
        JNZ     DECAD
HERE:   JMP     HERE
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)	Contenuti di Memoria (Esadec.)
00	LDA 30H	3A
01		30
02		00
03	MOV B,A	47
04	LXI H,41H	21
05		41
06		00
07	LXI D,61H	11
08		61
09		00
0A	ANA A	A7
0B	DECAD: LDAX D	1A
0C	ADC M	8E
0D	DAA	27
0E	MOV M,A	77
0F	INX D	13
10	INX H	23
11	DCR B	05
12	JNZ DECAD	C2
13		0B
14		00
15	HERE: JMP HERE	C3
16		15
17		00

L'istruzione di Aggiustamento Decimale (DAA) impiega i bit Carry e Carry Ausiliario per correggere le seguenti situazioni:

AGGIUSTAMENTO DECIMALE

1) La somma di due digit cade tra 10 e 15 compreso. In questo caso occorre aggiungere sei alla somma per avere il risultato corretto, cioè:

```
          0101  (5)
         +1000  (8)
          ----
          1101  (D)
         +0110
     ---------
     0011 0011  (BCD 13 che è il valore esatto)
```

2) La somma di due digit è maggiore o uguale a 16. In questo caso il risultato è proprio un digit BCD ma inferiore di sei di quello che dovrebbe essere cioè:

```
          1000  (8)
         +1001  (9)
     ---------
     0001 0001  (BCD 11)
         +0110
     ---------
     0001 0111  (BCD 17 che è il valore esatto)
```

Quindi occorre aggiungere sei in entrambe le situazioni. Comunque il caso 1 può essere riconosciuto dal fatto che la somma non è un digit BCD, cioè si trova tra 10 e 15 (oppure A ed F esadecimale). Invece il caso 2 si può riconoscere solo dal fatto che il Carry (digit più significativo) od il Carry Ausiliario (digit meno significativo) sono stati posti uguali ad 1 essendo il risultato un numero BCD valido. DAA è la sola istruzione che fa uso del Carry Ausiliario.

Questa procedura può sommare numeri decimali (BCD) di qualunque lunghezza. Qui sono richiesti quattro bit binari per ogni cifra decimale per cui la precisione di dieci cifre richiede:

$$10 \times 4 = 40 \text{ bit}$$

in contrapposizione ai 33 bit del caso binario. Sostanzialmente si tratta di cinque parole di otto bit invece di quattro. La procedura decimale richiede inoltre una maggiore lunghezza per parola a causa dell'ulteriore istruzione DAA.

Moltiplicazione Binaria ad 8 Bit

Scopo: Moltiplica il numero di 8 bit della locazione di memoria 40 con l'analogo della locazione 41. Posiziona gli otto bit meno significativi del risultato nella locazione 42 e gli 8 bit più significativi nella locazione 43.

Problemi Campione:

```
a.        (40)  =  03
          (41)  =  05
     Risultato  =  (42) = 0F
                   (43) = 00
                   cioè 3 X 5 = 15
b.        (40)  =  6F
          (41)  =  61
     Risultato  =  (42) = 0F
                   (43) = 2A
                   cioè 111 X 97 = 10,767
```

È possibile eseguire la moltiplicazione su un computer allo stesso modo di quella manuale. Poichè i numeri sono binari il problema si riduce alla moltiplicazione per 0 o per 1; moltiplicando per zero, ovviamente, si ottiene zero come risultato mentre la moltiplicazione per 1 origina lo stesso numero con cui si è partiti. Così ogni fase di una moltiplicazione binaria può essere ricondotta alla seguente operazione:

ALGORITMO DELLA MOLTIPLICAZIONE

Se il bit corrente del moltiplicatore è 1 somma il moltiplicando al prodotto precedente.

Rimane solo il problema di assicurarsi che ogni cosa venga allineata correttamente ogni volta che si esegue l'algoritmo. Le seguenti operazioni eseguono questo compito:

1) Sposta il moltiplicatore a sinistra di un bit in modo che il bit da esaminare è posizionato nel Carry.
2) Sposta il prodotto a sinistra di un bit in modo che l'addizione successiva è allineata correttamente.

Il processo completo per la moltiplicazione binaria è il seguente:

FASE 1 — Inizializzazione:
PRODOTTO = 0
CONTATORE = 8
FASE 2 — Sposta PRODOTTO così da allinearlo correttamente:
PRODOTTO = 2 X PRODOTTO (LSB = 0)
FASE 3 — Sposta MOLTIPLICATORE cosicchè il bit vada al CARRY.
MOLTIPLICATORE = 2 X MOLTIPLICATORE
FASE 4 — Somma MOLTIPLICANDO al PRODOTTO se CARRY è 1:
Se CARRY = 1, PRODOTTO = PRODOTTO + MOLTIPLICANDO
FASE 5 — Decrementa il CONTATORE e CONTROLLA per zero:
CONTATORE = CONTATORE −1
Se CONTATORE = 0, va alla FASE 2

Nel caso del problema campione b, dove il moltiplicatore è 61 (esadecimale) ed il moltiplicando è 6F (esadecimale) il processo lavora come segue:

Inizializzazione:

PRODOTTO	0000
MOLTIPLICATORE	61
MOLTIPLICANDO	6F
CONTATORE	08

Dopo la Prima iterazione delle FASI 2 a 5:

PRODOTTO	0000
MOLTIPLICATORE	C2
MOLTIPLICANDO	67
CONTATORE	07
CARRY DAL MOLTIPLICATORE	0

Dopo la seconda iterazione:

PRODOTTO	006F
MOLTIPLICATORE	84
MOLTIPLICANDO	6F
CONTATORE	06
CARRY DAL MOLTIPLICATORE	1

Dopo la terza iterazione:

PRODOTTO	014D
MOLTIPLICATORE	08
MOLTIPLICANDO	6F
CONTATORE	05
CARRY DAL MOLTIPLICATORE	1

Dopo la quarta iterazione:

PRODOTTO	029A
MOLTIPLICATORE	10
MOLTIPLICANDO	6F
CONTATORE	04
CARRY DAL MOLTIPLICATORE	0

Dopo la quinta iterazione:

PRODOTTO	0534
MOLTIPLICATORE	20
MOLTIPLICANDO	6F
CONTATORE	03
CARRY DAL MOLTIPLICATORE	0

Dopo la sesta iterazione:

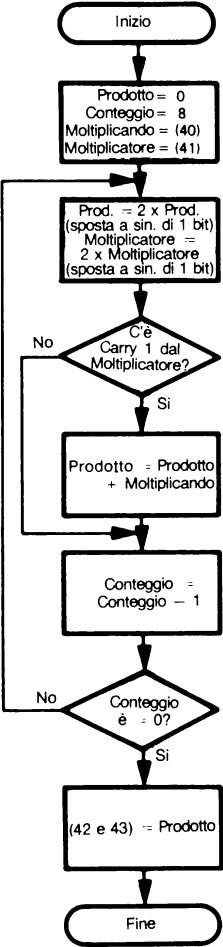
PRODOTTO	0A68
MOLTIPLICATORE	40
MOLTIPLICANDO	6F
CONTATORE	02
CARRY DAL MOLTIPLICATORE	0

Dopo la settima iterazione:

PRODOTTO	14D0
MOLTIPLICATORE	80
MOLTIPLICANDO	6F
CONTATORE	01
CARRY DAL MOLTIPLICATORE	0

Dopo l'ottava iterazione:

PRODOTTO	2A0F
MOLTIPLICATORE	00
MOLTIPLICANDO	6F
CONTATORE	00
CARRY DAL MOLTIPLICATORE	1

Diagramma di Flusso:

Programma Sorgente:

```
        LXI    H,40H
        MOV    E,M      ;ACCETTA MOLTIPLICANDO
        MVI    D,0      ;ESTENDI A 16 BIT
        INX    H
        MOV    A,M      ;ACCETTA MOLTIPLICATORE
        LXI    H,0      ;PRODOTTO = 0
        MVI    B,8      ;CONTEGGIO = 8
MULT:   DAD    H        ;PRODOTTO = PRODOTTO x 2
        RAL
        JNC    CHCNT    ;IL CARRY DEL MOLTIPLICATORE È 1?
        DAD    D        ;SÌ, PRODOTTO = PRODOTTO + MOLTIPLICANDO
CHCNT:  DCR    B
        JNZ    MULT
        SHLD   42H      ;CONSERVA IL PRODOTTO IN MEMORIA
HERE:   JMP    HERE
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)	Contenuti di Memoria (Esadec.)
00	LXI H,40H	21
01		40
02		00
03	MOV E,M	5E
04	MVI D,0	16
05		00
06	INX H	23
07	MOV A,M	7E
08	LXI H,0	21
09		00
0A		00
0B	MVI B,8	06
0C		08
0D	MULT: DAD H	29
0E	RAL	17
0F	JNC CHCNT	D2
10		13
11		00
12	DAD D	19
13	CHCNT: DCR B	05
14	JNZ MULT	C2
15		0D
16		00
17	SHLD	22
18		42
19		00
1A	HERE: JMP HERE	C3
1B		1A
1C		00

Si noti che il moltiplicando deve essere esteso a 16 bit mediante l'azzeramento del registro D cosicchè esso può essere sommato al prodotto mediante l'istruzione DAD.

L'istruzione DAD H agisce come spostamento a sinistra logico a 16 bit per il prodotto a 16 bit.

In questo programma le istruzioni a 16 bit dell'8080 trattano dati piuttosto che indirizzi. LXI viene impiegata per inizializzare il prodotto, DAD per eseguire spostamenti a 16 bit ed addizioni, infine SHLD per immagazzinare il risultato. Occorre assicurarsi di estendere la quantità da 8 bit (come il moltiplicando) a 16 bit. Si noti che non è possibile lavorare a 16 bit contemporaneamente per l'indirizzamento e la manipolazione dati.

Accanto agli impieghi ovvi nei calcolatori e nei terminali di punti di vendita, la moltiplicazione è una parte chiave di quasi tutta l'elaborazione dei segnali e gli algoritmi di controllo. La velocità con cui le moltiplicazioni possono essere eseguite può determinare il vantaggio di una CPU nel controllo di processo, rivelazione di segnale ed analisi di segnale. Questo algoritmo impiega da 190 a 230 microsecondi per moltiplicare su una CPU 8080 con un clock di 2 MHZ. Il tempo esatto dipende dal numero di bit 1 presenti nel moltiplicatore. Altri algoritmi possono essere in grado di ridurre il tempo medio di esecuzione ma 200 microsecondi rimarranno un tipico tempo di esecuzione per una moltiplicazione software.

Divisione Binaria ad 8 Bit

Scopo: Divide il numero generico a 16 bit delle locazioni di memoria 40 e 41 (bit più significativi in 41) con il numero generico ad 8 bit della locazione 42. I numeri sono normalizzati in modo che 1) i bit più significativi sia del dividendo che del divisore sono zero e 2) il numero della locazione di memoria 42 è maggiore di quello della locazione 41 cioè il quoziente può essere contenuto in 8 bit. Immagazzina il quoziente nella locazione di memoria 43 ed il resto nella locazione 44.

Problemi Campione:

a. (40) = 40 (64 decimale)
(41) = 00
(42) = 08
Risultato = (43) = 08
(44) = 00
cioè 64/8 = 8

b. (40) = 6D (12,909 decimale)
(41) = 32
(42) = 47 (71 decimale)
Risultato = (43) = B5 (181 decimale)
(44) = 3A (58 decimale)
cioè 12,909/71 = 181 come resto 58.

ALGORITMO DELLA DIVISIONE

È possibile eseguire la divisione sul computer esattamente come si eseguirebbe con carta e penna cioè impiegando sottrazioni di decisione. Poichè i numeri sono binari il solo problema è se il bit del quoziente è 0 o 1 cioè se il divisore può essere sottratto oppure no da quello che è a sinistra del dividendo. Ogni fase di una divisione binaria può essere ricondotta alla seguente operazione:

> Se il divisore può essere sottratto dagli 8 bit più significativi del dividendo senza un prestito il corrispondente bit del quoziente è 1; altrimenti è 0.

Rimane soltanto il problema del corretto allineamento del dividendo e del quoziente. Si può ottenere questo spostando in modo logico a sinistra di un bit il dividendo ed il quoziente prima di ogni sottrazione di decisione. Il dividendo ed il quoziente possono condividersi un registro a 16 bit poichè la procedura azzera un bit del dividendo nello stesso tempo di determinazione di un bit del quoziente.

Il processo completo per la divisione binaria è:

FASE 1 — Inizializzazione
QUOZIENTE = 0
CONTEGGIO = 8
FASE 2 — Sposta DIVIDENDO e QUOZIENTE per allinearli correttamente:
DIVIDENDO = 2 x QUOZIENTE
QUOZIENTE = 2 x QUOZIENTE
FASE 3 — Esegue la SOTTRAZIONE di decisione. Se non è necessario un PRESTITO somma 1 al QUOZIENTE:
Se gli 8 MSB del DIVIDENDO ≥ DIVISORE allora MSB del DIVIDENDO = MSB del DIVIDENDO - DIVISORE
QUOZIENTE = QUOZIENTE + 1
FASE 4 — Decrementa il contatore e controlla per zero:
CONTEGGIO = CONTEGGIO −1
Se CONTEGGIO = 0 va alla FASE 2
RESTO = 8 MSB del DIVIDENDO

Nel caso del problema campione b, dove il dividendo è 326D (esadec.) ed il divisore è 47 (esadec.) il processo lavora come segue:

Inizializzazione:

DIVIDENDO	326D
DIVISORE	47
QUOZIENTE	00
CONTEGGIO	00

Dopo la prima iterazione dalla FASE 2 alla 4:
(Si noti che il dividendo viene spostato prima della sottrazione di decisione)

DIVIDENDO	1DDA
DIVISORE	47
QUOZIENTE	01
CONTEGGIO	07

Dopo la seconda iterazione dalla FASE 2 alla 4:

DIVIDENDO	3BB4
DIVISORE	47
QUOZIENTE	02
CONTEGGIO	06

Dopo la terza iterazione:

DIVIDENDO	3068
DIVISORE	47
QUOZIENTE	05
CONTEGGIO	05

Dopo la quarta iterazione:

DIVIDENDO	19D0
DIVISORE	47
QUOZIENTE	0B
CONTEGGIO	04

Dopo la quinta iterazione:

DIVIDENDO	33A0
DIVISORE	47
QUOZIENTE	16
CONTEGGIO	03

Dopo la sesta iterazione:

DIVIDENDO	2040
DIVISORE	47
QUOZIENTE	2D
CONTEGGIO	02

Dopo la settima iterazione:

DIVIDENDO	4080
DIVISORE	47
QUOZIENTE	5A
CONTEGGIO	01

Dopo l'ottava iterazione:

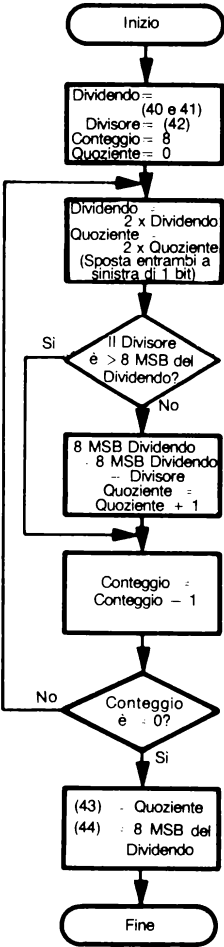
DIVIDENDO	3A00
DIVISORE	47
QUOZIENTE	B5
CONTEGGIO	00

Per cui il quoziente è B5 ed il resto è 3A.

Gli MSB del dividendo e del divisore sono assunti essere zero per semplificare i calcoli (lo spostamento precedente la sottrazione di decisione potrebbe altrimenti posizionare gli MSB del dividendo nel Carry). I problemi che non soddisfano le caratteristiche richieste possono essere riformulati rimuovendo quelle parti del quoziente che potrebbero fare uscire da una parola di 8 bit. Per esempio:

$$\frac{1024}{3} = \frac{400\ (\text{Es.})}{3} = 100 + \frac{100\ (\text{Es.})}{3}$$

L'ultima divisione è ora in forma appropriata. In casi particolari può essere necessaria un'ulteriore divisione.

Diagramma di Flusso:

Programma Sorgente:

```
        LHLD    40H     ;ACCETTA DIVIDENDO
        LDA     42H     ;ACCETTA DIVISORE
        MOV     C,A
        MVI     B,8     ;CONTEGGIO = 8
DIV:    DAD     H       ;SPOSTA DIVIDENDO, QUOZIENTE
        MOV     A,H     ;LA PARTE PIÙ SIGNIFICATIVA DEL DIVIDENDO
                        ;È ≥ DIVISORE
        SUB     C
        JC      CNT     ;NO, VA ALLA FASE SUCCESSIVA
        MOV     H,A     ;SÌ, SOTTRAI DIVISORE
        INR     L       ;E SOMMA 1 AL QUOZIENTE
CNT:    DCR     B
        JNZ     DIV
        SHLD    43H     ;IMMAGAZZINA QUOZIENTE, RESTO
HERE:   JMP     HERE
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)	Contenuti di Memoria (Esadec.)
00	LHLD 40H	2A
01		40
02		00
03	LDA 42H	3A
04		42
05		00
06	MOV C,A	4F
07	MVI B,8	06
08		08
09	DIV: DAD H	29
0A	MOV A,H	7C
0B	SUB C	91
0C	JC CNT	DA
0D		11
0E		00
0F	MOV H,A	67
10	INR L	2C
11	CNT: DCR B	05
12	JNZ DIV	C2
13		09
14		00
15	SHLD 43H	22
16		43
17		00
18	HERE: JMP HERE	C3
19		18
1A		00

I registri H ed L conservano sia il dividendo che il quoziente. Il quoziente semplicemente sostituisce il dividendo nel Registro L mediante spostamento logico a sinistra del dividendo stesso.

Una sottrazione ad 8 bit è necessaria (più il movimento di alcuni registri) poichè non esiste un modo semplice per eseguire una sottrazione o confronto a 16 bit. Comunque, DAD H esegue uno spostamento a sinistra a 16 bit del dividendo e quoziente.

L'istruzione INR L pone ad 1 il bit meno significativo del quoziente poichè DAD H ha precedentemente azzerato questo bit.

La divisione è impiegata nei calcolatori, terminali, controllo errori di comunicazioni, controllo algoritmi ed in molte altre applicazioni.

L'algoritmo impiega da 200 a 250 microsecondi per dividere su un Intel 8080 con un clock di 2 MHz. Il tempo esatto dipende dal numero di bit 1 del quoziente. Altri algoritmi possono ridurre il tempo medio di esecuzione ma 250 microsecondi sarà ancora un tempo tipico per una divisione software.

Numeri Self-Checking

Somma di Doppio con Doppio, Mod 10

Scopo: Calcola il digit checksum da una stringa di digit BCD. La lunghezza della stringa di digit (numero di parole) è indicata nella locazione di memoria 30; la stringa di digit (2 digit BCD per ogni parola) inizia nella locazione 41. Calcola infine il digit checksum mediante la tecnica SOMMA DI DOPPIO CON DOPPIO MOD 10 e lo immagazzina nella locazione 40 (vedere J.R. Herr, «Self-Checking Number Systems», COMPUTER DESIGN, Giugno 1974, pp. 85-91).

La tecnica SOMMA DI DOPPIO CON DOPPIO MOD 10 lavora come segue:

1) Azzera checksum all'inizio.
2) Moltiplica il digit iniziale per due e somma il prodotto a checksum.
3) Somma il successivo digit a checksum.
4) Ripete il processo fino ad aver impiegato tutti i digit.
5) Il digit meno significativo di checksum è il digit self-checking.

I digit self-checking vengono normalmente aggiunti ai numeri di identificazione delle carte di credito, etichette di inventore, bagagli, pacchi che vengono maneggiati da sistemi computerizzati. Essi possono anche essere impiegati nell'indagine di messaggi, identificazione di schedari ed altre applicazioni. Lo scopo del digit è la minimizzazione di errori di ingresso come trasposizione di digit (69 invece di 96), scorrimento di digit (7260 al posto di 3726), omissione di 1 da un digit (65 al posto di 64), ecc. È possibile controllare automaticamente il numero self-checking per operare correzioni prima dell'ingresso ed eliminare immediatamente molti errori.

L'analisi dei metodi self-checking è abbastanza complessa. Per esempio una semplice checksum non sarà di aiuto con trasposizione (per esempio $4 + 9 = 9 + 4$). L'algoritmo SOMMA DI DOPPIO con DOPPIO trova gli errori di trasposizione (per esempio $2 \times 4 + 9 = 17 \neq 2 \times 9 + 4$) ma non rivela altri errori (per esempio $2 \times 5 + 3$ ha lo stesso digit meno significativo di $2 \times 0 + 3$ cosicchè il metodo non troverà questo errore).

Per esempio se la stringa di digit è:

549321

il risultato è:

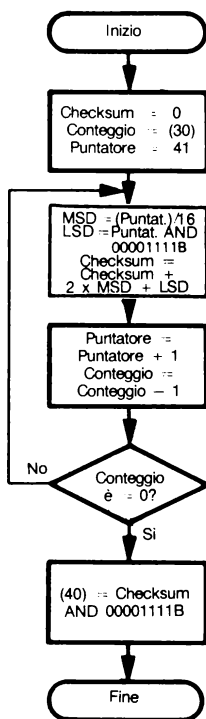
Checksum $= 5 X 2 + 4 + 9 X 2 + 3 + 2 X 2 + 1 = 40$
Digit Self-Checking $=$ 0 (digit meno significativo di checksum)

Si noti che un ingresso errato come 543921 produrrebbe un diverso digit self-checking (4) mentre un ingresso errato come 043921 non sarebbe rivelabile.

Problemi Campione:

a. (30) = 03
(41) = 36
(42) = 68
(43) = 51
Risultato = Checksum = 3 x 2 + 6 + 6 x 2 + 8 + 5 x 2 + 1 = 43
(40) = 03

b. (30) = 04
(41) = 50
(42) = 29
(43) = 16
(44) = 83
Risultato = Checksum = 5 x 2 + 0 + 2 x 2 + 9 + 1 x 2 + 6 + 8 x 2 + 3 = 50
(40) = 00

Diagramma di Flusso:

Programma Sorgente:

```
        LDA     30H         ;CONTEGGIO = LUNGHEZZA DELLA STRINGA
                            ;(UN BYTE)
        MOV     B,A
        MVI     C,0         ;CHECKSUM = 0
        LXI     H,41H       ;PUNTA ALL'INIZIO DELLA STRINGA
CHDIG:  MOV     A,M         ;ACCETTA DUE DIGIT BCD
        MOV     D,A         ;CONSERVA LA COPIA
        RAR                 ;ACCETTA MS MEDIANTE SCORRIMENTO
                            ;E MASCHERATURA
        RAR
        RAR
        RAR
        ANI     00001111B   ;MASCHERA OFF PER MSD
        ADD     A           ;DUPLICA MSD
        DAA                 ;CONSERVA IL DECIMALE
        ADD     C           ;SOMMA 2 X MSD ALLA CHECKSUM
        DAA
        MOV     C,A
        MOV     A,D
        ANI     00001111B   ;MASCHERA OFF PER LSD
        ADD     C           ;SOMMA LSD A CHECKSUM
        DAA
        MOV     C,A
        INX     H
        DCR     B
        JNZ     CHDIG
        ANI     00001111B   ;MASCHERA OFF PER IL DIGIT SELF-CHECKING
        STA     40H         ;CONSERVA IL DIGIT SELF-CHECKING
HERE:   JMP     HERE
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)			Contenuti di Memoria (Esadec.)
00		LDA	30H	3A
01				30
02				00
03		MOV	B,A	47
04		MVI	C,0	0E
05				00
06		LXI	H,41H	21
07				41
08				00
09	CHDIG:	MOV	A,M	7E
0A		MOV	D,A	57
0B		RAR		1F
0C		RAR		1F
0D		RAR		1F
0E		RAR		1F
0F		ANI	00001111B	E6
10				0F
11		ADD	A	87
12		DAA		27
13		ADD	C	81
14		DAA		27
15		MOV	C,A	4F
16		MOV	A,D	7A
17		ANI	00001111B	E6
18				0F
19		ADD	C	81
1A		DAA		27
1B		MOV	C,A	4F
1C		INX	H	23
1D		DCR	B	05
1E		JNZ	CHDIG	C2
1F				09
20				00
21		ANI	00001111B	E6
22				0F
23		STA	40H	32
24				40
25				00
26	HERE:	JMP	HERE	C3
27				26
28				00

I digit sono rimossi mediante spostamento e mascheratura. Quattro spostamenti a destra sono richiesti per ottenere il digit più significativo.

Un Aggiustamento Decimale (DAA) deve seguire ogni addizione per prendere il risultato corretto. Una singola istruzione DAA dopo una serie di addizioni non esegue il compito (si provi in questo programma).

La duplicazione del MSD comprende l'addizione con sè stesso e poi l'esecuzione dell'Aggiustamento Decimale (DAA).

I riporti dalla somma decimale vengono ignorati poichè la procedura impiega solo il digit meno significativo di checksum.

PROBLEMI

1) Sottrazione a Precisione Multipla

Scopo: Sottrae due numeri a parola multipla. La lunghezza dei numeri è indicata nella locazione di memoria 30, gli stessi numeri iniziano (prima i bit meno significativi) nelle locazioni di memoria 41 e 61 rispettivamente e la differenza sostituisce il numero iniziante nella locazione 41. Sottrae il numero iniziante in 61 da quello iniziante in 41.

Problema Campione:

```
      (30)  =  04
      (41)  =  C3
      (42)  =  A7
      (43)  =  5B
      (44)  =  2F
      (61)  =  B8
      (62)  =  35
      (63)  =  DF
      (64)  =  14
 Risultato  =  (41) = 0B
               (42) = 72
               (43) = 7C
               (44) = 1A
      cioè     2F5BA7C3
            +  14DF35B8
               --------
               1A7C720B
```

2) Sottrazione Decimale

Scopo: Sottrae due numeri decimali (BCD) a parola multipla. La lunghezza dei numeri è indicata nella locazione di memoria 30, gli stessi numeri iniziano (prima i bit meno significativi) nelle locazioni di memoria 41 e 61 rispettivamente e la differenza sostituisce il numero iniziante in 41. Viene sottratto il numero iniziante in 61 da quello iniziante in 41.

Problema Campione:

```
      (30)  =  04
      (41)  =  85
      (42)  =  19
      (43)  =  70
      (44)  =  36
      (61)  =  59
      (62)  =  34
      (63)  =  66
      (64)  =  12
 Risultato  =  (41) = 26
               (42) = 85
               (43) = 03
               (44) = 24
      cioè     36701985
            +  12663459
               --------
               24038526
```

Questo servirà da suggerimento:

$$X - Y = X + 99 - Y + \overline{PRESTITO}$$

dove X ed Y sono 2 digit ciascuno provenienti da una stringa e PRESTITO è il prestito dai digit meno significativi. Trovando 99 - Y non è un problema Y è sempre minore od uguale a 99. Si ricordi però che il ruolo del CARRY è opposto rispetto a quello usuale (perchè?).

3) Moltiplicazione Binaria di 8 Bit con 16 Bit

Scopo: Moltiplica il generico numero a 16 bit delle locazioni di memoria 40 e 41 (bit più significativi in 41) con il numero generico ad 8 Bit della locazione 42. Immagazzina il risultato nelle locazioni da 43 a 45 con i bit più significativi nella 45.

Problemi Campione:

a.
(40) = 03
(41) = 00
(42) = 05
Risultato = (43) = 0F
(44) = 00
(45) = 00
cioè 3 x 5 = 15

b.
(40) = 6F (29,295 decimale)
(41) = 72
(42) = 61 (97 decimale)
Risultato = (43) = 0F
(44) = 5C
(45) = 2B
cioè 29,295 x 97 = 2,841,615

Procede esattamente come nell'esempio della Moltiplicazione binaria ad 8 Bit ma conserva gli 8 Bit più significativi del prodotto nell'Accumulatore cioè li fa scorrere in questo ultimo in corrispondenza dello scorrimento del moltiplicatore.

4) Divisione Binaria Con Segno

Scopo: Divide il numero con segno a 16 bit delle locazioni di memoria 40 e 41 (bit più significativi In 41) con il numero con segno ad 8 bit della locazione 42. I numeri sono normalizzati cosicchè la grandezza della locazione di memoria 42 è maggiore di quella della locazione 41, cioè il quoziente può essere contenuto in 8 bit. Immagazzina il quoziente (con segno) nella locazione di memoria 43 ed il resto (sempre positivo) nella locazione 44.

Problemi Campione:

a.
(40) = C0 (–64)
(41) = FF
(42) = 08
Risultato = (43) = F8 (–8) quoziente
(44) = 00 (0) resto

b.
(40) = 93 (–4,717)
(41) = ED
(42) = 47 (71 decimale)
Risultato = (43) = BD (–67) decimale
(44) = 2B (+40) decimale

Determina il segno del risultato, esegue una divisione senza segno ed aggiusta il quoziente ed il resto in forma opportuna.

5) Numeri Self-Checking ALLINEATI 1, 3, 7, MOD 10

Scopo: Calcola il digit checksum da una stringa di digit BCD. La lunghezza della stringa di digit (numero di parole) è indicata nella locazione di memoria 30; la stringa di digit (2 digit BCD per ogni parola) inizia nella locazione di memoria 41. Calcola il digit checksum mediante il metodo ALLINEATI 1, 3, 7 MOD 10 e lo immagazzina nella locazione di memoria 40.

La tecnica ALLINEATI 1, 3, 7 MOD 10 lavora come segue:

1) Azzera checksum all'inizio.
2) Somma a checksum il digit iniziale.
3) Moltiplica il successivo digit per 3 e somma il prodotto a checksum.
4) Moltiplica il successivo digit per 7 e somma il prodotto a checksum.
5) Continua il processo (Fasi dalla 2 alla 4) fino ad aver impiegato tutti i digit.
6) Il digit self-checking è quello meno significativo della checksum.

Per esempio se la stringa di digit è:

549321

il risultato è:

Checksum = 5 + 3 X 4 + 7 X 9 + 3 + 3 X 2 + 7 X 1 = 96
Digit Self-Checking = 6

Problemi Campione:

a.
(30) = 03
(41) = 36
(42) = 68
(43) = 51
Risultato = Checksum = 3 + 3 X 6 + 7 X 6 + 8 + 3 X 5 + 7 X 1 = 93
(40) = 03

b.
(30) = 04
(41) = 50
(42) = 29
(43) = 16
(44) = 83
Risultato = Checksum = 5 + 3 X 0 + 7 X 2 + 9 + 3 X 1 + 7 X 6 + 8 + 3 X 3 = 90
(40) = 00

Si noti che 7 = 2 X 3 + 1 e 3 = 2 X 1 + 1 cosicchè la formula $M_i = 2 \times M_{i-1} + 1$ può essere usata per ottenere un fattore moltiplicativo quello successivo.

Capitolo 9
TABELLE E LISTE

Le Tabelle e le liste sono due strutture base di dati impiegate in tutti i sistemi computer. Si è già visto l'impiego di tabelle per l'esecuzione di operazioni aritmetiche e conversioni di codice. Le tabelle possono anche identificare o rispondere a comandi ed istruzioni, allineare dati, fornire l'accesso ad archivi o dischi, definire il significato di tasti o interruttori e scegliere sequenze di istruzioni. Le liste generalmente hanno una struttura più semplice delle tabelle. Le liste possono registrare compiti che il processore deve eseguire, messaggi d'uscita o dati che il processore deve conservare oppure condizioni che sono cambiate o che, potrebbero essere ispezionate. Le tabelle sono un mezzo semplice per prendere decisioni o risolvere problemi perchè non sono necessari nè calcoli nè funzioni logiche. Il problema si riduce quindi ad organizzare la tabella cosicchè l'ingresso appropriato sia facile da trovare. Le liste permettono l'esecuzione di compiti multipli, la preparazione di risultati multipli e la costruzione di archivi dati correlati (o data base). I problemi principali sono come aggiungere elementi alla lista e come cancellarli.

ESEMPI

Aggiunta di Ingresso alla Lista

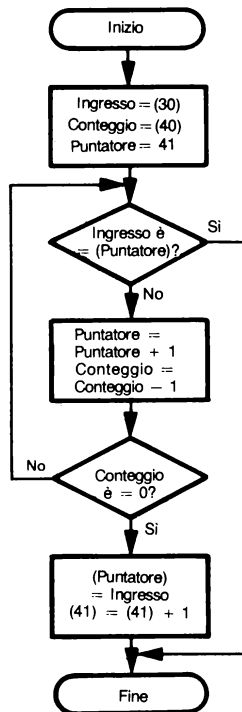
Scopo: Aggiunge i contenuti della locazione di memoria 30 alla lista se non è già presente. La lunghezza della lista è indicata nella locazione di memoria 40 e la stessa lista inizia dalla locazione 42.

Problemi Campione:

a.
(30) = 6B
(40) = 04
(41) = 37
(42) = 61
(43) = 28
(44) = 1D
Risultato = (40) = 05
(45) = 6B

L'ingresso è aggiunto alla lista poichè esso non è già presente. La lunghezza della lista è aumentata di 1.

b.
(30) = 6B
(40) = 04
(41) = 37
(42) = 6B
(43) = 28
(44) = 1D
Risultato = Invariato poichè l'ingresso è già presente nella lista.

Diagramma di Flusso:

Programma Sorgente:

```
        LXI   41H
        MOV   B,M     ;CONTEGGIO = LUNGHEZZA DELLA LISTA
        INX   H       ;PUNTA ALL'INIZIO DELLA LISTA
        LDA   30H     ;ACCETTA INGRESSO
SRLST:  CMP   M       ;L'INGRESSO È = ELEMENTO IN LISTA?
        JZ    DONE    ;SÌ, SALTA ALLA FINE
        INX   H       ;NO, VA A VEDERE IL PROSSIMO ELEMENTO
        DCR   B
        JNZ   SRLST   ;TUTTI GLI ELEMENTI SONO STATI ESAMINATI?
        MOV   M,A     ;SÌ, AGGIUNGE L'INGRESSO ALLA LISTA
        LXI   H,41H
        INR   M       ;SOMMA 1 ALLA LUNGHEZZA DELLA LISTA
DONE:   JMP   DONE
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)			Contenuti di Memoria (Esadec.)
00		LXI	H,41H	21
01				41
02				00
03		MOV	B,M	46
04		INX	H	23
05		LDA	30H	3A
06				30
07				00
08	SRLST:	CMP	M	BE
09		JZ	DONE	CA
0A				16
0B				00
0C		INX	H	23
0D		DCR	B	05
0E		JNZ	SRLST	C2
0F				08
10				00
11		MOV	M,A	77
12		LXI	H,41H	21
13				41
14				00
15		INR	M	34
16	DONE:	JMP	DONE	C3
17				16
18				00

Il programma non opera se la lunghezza della lista è zero. Si potrebbe evitare questo problema controllando inizialmente la lunghezza. La procedura di inizializzazione sarebbe quindi:

```
        LXI   H,42H
        MOV   B,M      ;CONTEGGIO = LUNGHEZZA DELLA LISTA
        SUB   A
        ORA   B        ;PONE FLAG DAL CONTEGGIO
        LDA   30H      ;ACCETTA INGRESSO
        JNX   H        ;PUNTA ALL'INIZIO DELLA LISTA
        JZ    ADELM    ;AGGIUNGE IL PRIMO INGRESSO SE LA LISTA È VUOTA
        —
        —
        —
ADELM:  MOV   M,A      ;AGGIUNGI INGRESSO ALLA LISTA
        —
        —
        —
```

La procedura:

```
        LXI   H,ADDR
        INR   M
```

è un metodo veloce per sommare 1 ad un contatore posizionato nella locazione di memoria ADDR. Impiegando DCR M in modo simile si sottrae 1 dal contatore. MVI M, CONST può posizionare un valore di partenza (come uno zero) nel contatore. Le locazioni di memoria dovrebbero, naturalmente, essere usate solo per contatori qualora non fossero disponibili i registri.

HASHING

Chiaramente questo metodo di aggiungere elementi è molto inefficiente se la lista è lunga. Si potrebbe migliorare la procedura limitando la ricerca a parte della lista o mediante ordinamento della lista stessa. Si potrebbe limitare la ricerca impiegando l'ingresso per scegliere il punto di partenza della lista. Questo metodo è chiamato hashing ed è simile alla selezione della pagina iniziale in un dizionario od elenco sulla base della prima lettera di un ingresso. Si potrebbe ordinare la lista mediante valori numerici. La ricerca potrebbe poi terminare quando i valori della lista sono andati oltre l'ingresso (i valori potrebbero essere maggiori che minori in dipendenza della tecnica di ordinamento impiegata). La difficoltà connessa con questo metodo è che un nuovo ingresso dovrebbe essere inserito al posto giusto e così tutti gli ingressi successivi dovrebbero essere spostati in basso nella lista.

Il programma potrebbe essere strutturato per usare due tabelle. Una tabella potrebbe fornire un punto iniziale di ricerca nell'altra tabella, per esempio il punto di ricerca potrebbe essere basato sul digit a 4 bit più oppure meno significativo dell'ingresso.

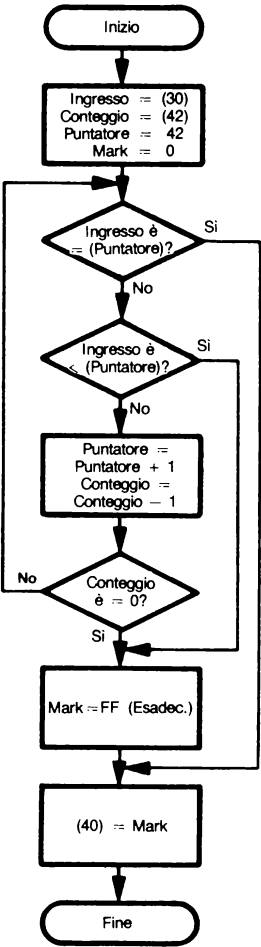
Se ogni ingresso era più lungo di una parola una procedura di confronto di stringhe di caratteri (pattern-matching) potrebbe essere necessaria come nell'esempio considerato in precedenza. Si dovrebbe fare molta attenzione ad allineare il successivo ingresso correttamente se un confronto non è soddisfatto, cioè saltare l'ultima parte dell'ingresso attuale una volta trovata una differenza.

Controllo di una lista ordinata

Scopo: Controlla i contenuti della locazione di memoria 30 per vedere se si trovano in una lista ordinata. La lunghezza della lista è indicata nella locazione di memoria 41; la stessa lista inizia nella locazione 42 e consiste di numeri binari senza segno in ordine crescente. Se i contenuti della locazione 30 si trovano nella lista azzera la locazione di memoria 40; diversamente pone la locazione 40 ad FF (esadec.).

Problemi Campione:

a.	(30)	=	6B
	(41)	=	04
	(42)	=	37
	(43)	=	55
	(44)	=	7D
	(45)	=	A1
	Risultato	=	(40) = FF
b.	(30)	=	6B
	(41)	=	04
	(42)	=	37
	(43)	=	55
	(44)	=	6B
	(45)	=	A1
	Risultato	=	(40) = 00

Diagramma di Flusso:

Programma Sorgente:

```
        LXI   H,41H
        MOV   B,M      ;CONTEGGIO = LUNGHEZZA DELLA LISTA
        INX   H        ;PUNTO DI INIZIO DELLA LISTA
        MVI   C,0      ;MARK = 0 PER IN LISTA
        LDA   30H      ;ACCETTA INGRESSO
SRLST:  CMP   M        ;L'INGRESSO È = ELEMENTO IN LISTA?
        JZ    DONE     ;SÌ, SALTA ALLA FINE
        JC    NOTIN    ;INGRESSO NON IN LISTA SE < ELEMENTO
        INX   H        ;VA A VEDERE L'ELEMENTO SUCCESSIVO
        DCR   B
        JNZ   SRLST    ;TUTTI GLI ELEMENTI SONO STATI ESAMINATI?
NOTIN:  MVI   C,0FFH   ;SÌ, MARK = FF PER NON IN LISTA
DONE:   MOV   A,C      ;CONSERVA MARK = 00 OPPURE FF
        STA   40H
HERE:   JMP   HERE
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)	Contenuti di Memoria (Esadec.)
00	LXI H,41H	21
01		41
02		00
03	MOV B,M	46
04	INX H	23
05	MVI C,0	0E
06		00
07	LDA 30H	3A
08		30
09		00
0A	SRLST: CMP M	BE
0B	JZ DONE	CA
0C		18
0D		00
0E	JC NOTIN	DA
0F		16
10		00
11	INX H	23
12	DCR B	05
13	JNZ SRLST	C2
14		0C
15		00
16	NOTIN: MVI C,0FFH	0E
17		FF
18	DONE: MOV A,C	79
19	STA 40H	32
1A		40
1B		00
1C	HERE: JMP HERE	C3
1D		1C
1E		00

Il processo di ricerca qui è un po' diverso poichè gli elementi sono ordinati. Una volta trovato un elemento maggiore dell'ingresso la ricerca è inutile poichè tutti gli elementi successivi saranno ancora maggiori. Si può provare un esempio per convincersi come opera la procedura.

METODI DI RICERCA

Come nel problema precedente una tabella oppure altri metodi che scelgono un buon punto di partenza potrebbero accelerare la ricerca. Un metodo potrebbe essere quello di partire a metà e poi determinare quale metà potrebbe contenere l'ingresso, poi dividere questa metà in due parti uguali, ecc.. Questo metodo è chiamato «ricerca binaria» poichè divide le parti rimanenti della lista in due parti uguali ogni volta. Knuth descrive altre tecniche di ricerca nel libro: «The Art of Computer Programming, Volume III: Sorting And Searching», Addison-Wesley, Reading, Mass., 1973 — Knuth ha anche discusso la ricerca e l'hashing in modo più elementare in un articolo intitolato «Alorithms» (Vedere il numero di Aprile 1977 di Scientific American).

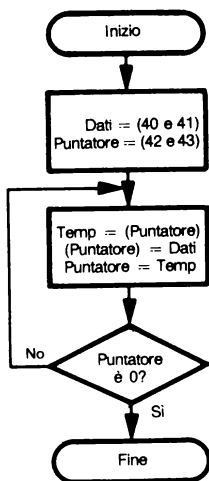
Questo algoritmo è un po' più lento di quello dell'esempio fatto sotto il titolo AGGIUNGE INGRESSO ALLA LISTA a causa del salto condizionato interno (JC NOTIN). Il tempo medio di esecuzione per questa semplice tecnica di ricerca aumenta linearmente con la lunghezza della lista mentre il tempo medio di esecuzione per la ricerca binaria è proporzionale al logaritmo della lunghezza della lista (il tempo di ricerca aumenta di una iterazione per ogni raddoppio della lunghezza della lista). Si noti che si potrebbe sostituire l'istruzione MVI, C, OFFH di label NOTIN con una istruzione DCR C e risparmiare quindi un byte del codice oggetto. Perchè è consentito fare questo?

Sostituzione di Una Catena di Indirizzi con Dati

Scopo: Sostituisce con dati ogni ingresso di una catena di indirizzi. I dati sono nelle locazioni di memoria 40 e 41 (MSB in 41). L'indirizzo della catena si trova nelle locazioni di memoria 42 e 43 (MSB in 43). Ogni ingresso nella catena è lungo due byte e punta all'indirizzo del successivo elemento a due byte della catena. L'ultimo elemento della catena contiene zero per indicare che non esistono elementi successivi.
Questo tipo di struttura dati (la catena) è impiegata per permettere reference dirette, cioè sono noti i simboli di reference prima che vengano definiti i dati. L'impiego più comune è in assemblatori e compilatori. In questo uso quando un simbolo, che non è ancora stato definito (per definito si intende stabilito mediante un EQU oppure impiegato come label ecc.), viene consultato, allora si opera un ingresso alla catena di tutte le reference per quel simbolo. Quando il simbolo è finalmente definito tutte le reference relative a quel simbolo sono quindi soddisfatte dall'elaborazione della catena.

Problema Campione:

```
          (40)  = 66 } dati
          (41)  = 00 }
          (42)  = 46 } indirizzo del primo elemento in lista
          (43)  = 00 }
          (46)  = 4D } indirizzo del secondo elemento in lista
          (47)  = 00 }
          (4D)  = 00 } fine della lista
          (4E)  = 00 }
     Risultato  = (46) = 66
                  (47) = 00
                  (4D) = 66
                  (4E) = 00
```

Diagramma di Flusso:

Programma Sorgente:

```
        LHLD    40H     ;ACCETTA E CONSERVA I DATI
        MOV     B,H
        MOV     C,L
        LHLD    42H     ;PUNTA ALL'INIZIO DELLA LISTA
CHAIN:  MOV     E,M     ;ACCETTA L'INDIRIZZO DELL'ELEMENTO
                        SUCCESSIVO
        MOV     M,C
        INX     H
        MOV     D,M
        MOV     M,B     ;SOSTITUISCE IL PUNTATORE CON DATI
        XCHG            ;NUOVO PUNTATORE
        MOV     A,H     ;IL NUOVO PUNTATORE È ZERO?
        ORA     L       ;SE COSÌ, PIÙ NESSUN ELEMENTO
        JNZ     CHAIN   ;SE NO, CONTINUA A PERCORRERE LA CATENA
HERE:   JMP     HERE
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)			Contenuti di Memoria (Esadec.)
00		LHLD	40H	2A
01				40
02				00
03		MOV	B,H	44
04		MOV	C,L	4D
05		LHLD	42H	2A
06				42
07				00
08	CHAIN:	MOV	E,M	5E
09		MOV	M,C	71
0A		INX	H	23
0B		MOV	D,M	56
0C		MOV	M,B	70
0D		XCHG		EB
0E		MOV	A,H	7C
0F		ORA	L	B5
10		JNZ	CHAIN	C2
11				08
12				00
13	HERE:	JMP	HERE	C3
14				13
15				00

Il processore 8080 è dotato di alcune istruzioni a 16 bit ma non un set. Non esiste l'indirizzamento di memoria implicato con dati a 16 bit come pure non esiste un modo per controllare se un numero a 16 bit è 0. Perciò occorre usare l'istruzione ad 8 bit MOV per ottenere il nuovo puntatore e riempire quello vecchio con dati. È necessario inoltre impiegare l'operazione di OR logico di due parti ad 8 bit per determinare se il puntatore a 16 bit è zero (si noti che sommando le due parti ad 8 bit il programma potrebbe non funzionare – perchè?).

L'istruzione XCHG, che scambia i contenuti della coppia di registri H ed L con D ed E, è conveniente come istruzione di trasferimento tra registri a 16 bit. XCHG sostituisce l'intera serie di istruzioni MOV poichè essa scambia due quantità a 16 bit. Si noti che non si può trasferire il puntatore immediatamente ad H ed L (cosa succede se si sostituisce MOV E,M con MOV L,M?).

Gli incatenamenti possono maneggiare liste che si trovano in locazioni di memoria non sequenziali. Ogni elemento deve comunque contenere l'indirizzo del successivo. Queste liste possono permettere ad un utente di cambiare variabili, di inserire definizioni in un programma oppure di creare un collegamento data base.

Classificazione ad 8-Bit

Scopo: Classifica un array di numeri senza segno in ordine decrescente. La lunghezza dell'array è indicata nella locazione di memoria 40 e lo stesso array inizia nella locazione di memoria 41.

Problema Campione:

(40) = 06
(41) = 2A
(42) = B5
(43) = 60
(44) = 3F
(45) = D1
(46) = 19
Risultato = (41) = D1
(42) = B5
(43) = 60
(44) = 3F
(45) = 2A
(46) = 19

Una semplice tecnica di classificazione lavora come segue:

FASE 1 – Azzera il flag INTER
FASE 2 – Esamina successivamente ogni coppia di numeri dell'array.
Se qualcuno è fuori dall'ordine li scambia e pone il flag INTER ad 1.
FASE 3 – Se INTER = 0, ritorna alla FASE 1

SEMPLICE ALGORITMO DI CLASSIFICAZIONE

INTER sarà posto ad 1 se qualsiasi coppia di numeri è fuori dall'ordine. Perciò se INTER = 0 l'array è in ordine corretto. La tecnica funziona come riportato in seguito in un caso semplice. Si supponga di voler classificare un array in ordine ascendente; l'array abbia quattro elementi – 08, 15, 03, 12.

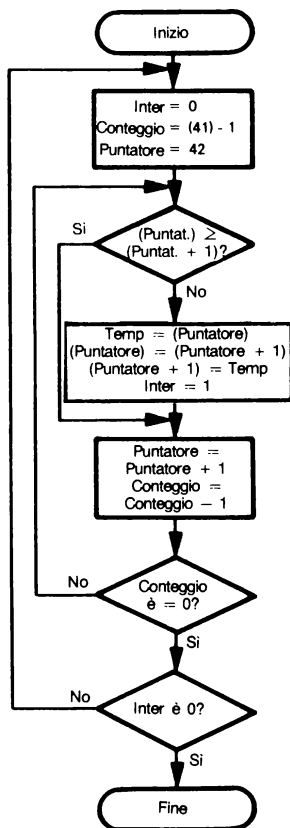
Prima iterazione:
FASE 1 – INTER = 0
FASE 2 – L'ordine finale dell'array è
15
08
12
03

Poichè la prima coppia (08, 15) e la terza (03, 12) sono cambiate è INTER = 1

Seconda iterazione:
FASE 1 – INTER = 0
FASE 2 – L'ordine finale dell'array è:
15
12
08
03

Poichè la seconda coppia (08, 12) è cambiata è INTER = 1

Terza iterazione:
FASE 1 — INTER = 0
FASE 2 — Gli elementi sono già in ordine cosicchè non sono necessari scambi ed INTER rimane zero.

Diagramma di Flusso:

Programma Sorgente:

```
SORT:    MVI   B,0      ;FLAG DI SCAMBIO = 0
         LXI   H,41H    ;CONTEGGIO = LUNGHEZZA DELL'ARRAY
         MOV   C,M
         DCR   C        ;NUMERO DI COPPIE = CONTEGGIO -1
         INX   H        ;PUNTA ALL'INIZIO DELL'ARRAY
PASS 1:  MOV   A,M      ;ACCETTA K-ESIMO ELEMENTO
         INX   H
         CMP   M        ;PARAGONA CON L'ELEMENTO (K + 1)-ESIMO
         JNC   CNT      ;NON SCAMBIA SE K-ESIMO ≥ (K + 1)-ESIMO
         MOV   D,M      ;SCAMBIA SE FUORI DALL'ORDINE
         MOV   M,A
         DCX   H
         MOV   M,D
         INX   H
         MVI   B,1      ;FLAG DI SCAMBIO = 1
CNT:     DCR   C        ;CONTO ALLA ROVESCIA
         JNZ   PASS 1
         DCR   B        ;IL FLAG DI SCAMBIO È 1?
         JZ    SORT     ;SÌ, ESEGUI UN ALTRO PASSO
HERE:    JMP   HERE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00	SORT:	MVI	B,0	06
01				00
02		LXI	H,41H	21
03				41
04				00
05		MOV	C,M	4E
06		DCR	C	0D
07		INX	H	23
08	PASS1:	MOV	A,M	7E
09		INX	H	23
0A		CMP	M	BE
0B		JNC	CNT	D2
0C				15
0D				00
0E		MOV	D,M	56
0F		MOV	M,A	77
10		DCX	H	2B
11		MOV	M,D	72
12		INX	H	23
13		MVI	B,1	06
14				01
15	CNT:	DCR	C	0D
16		JNZ	PASS1	C2
17				08
18				00
19		DCR	B	05
1A		JZ	SORT	CA
1B				00
1C				00
1D	HERE:	JMP	HERE	C3
1E				1D
1F				00

Qui è molto importante il caso di due elementi dell'array uguali. Il programma non dovrebbe eseguire uno scambio in questo caso perchè questo scambio dovrebbe ripetersi ad ogni passo. Il risultato potrebbe essere che ad ogni passo si pone ad 1 il flag di scambio producendo così un ciclo senza fine.

L'istruzione di Ramificazione Condizionata dell'8080 può essere limitante e lo è particolarmente in questo programma. A seguito dell'istruzione CMP M ci può essere JC, salta se (M) > (A), e JNC, salta se (M) $\leq$ (A). Non esistono istruzioni di salto con la condizione di uguaglianza invertita cioè (M) $\geq$ (A) ed (M) < (A). Perciò occorre prestare molta attenzione all'ordine delle operazioni.

L'indirizzamento di memoria implicato attraverso i registri H ed L è inopportuno in questo caso poichè il programma impiega coppie di elementi piuttosto che singoli elementi. Si noti che se l'8080 consentisse un'indicizzazione questo programma potrebbe essere molto più semplice, poichè i due elementi potrebbero essere riferiti con lo stesso indirizzo di base ma con indici diversi.

Prima di ogni passo di classificazione occorre assicurarsi di inizializzare il contatore, il puntatore ed il flag di scambio.

ALTRI METODI DI CLASSIFICAZIONE

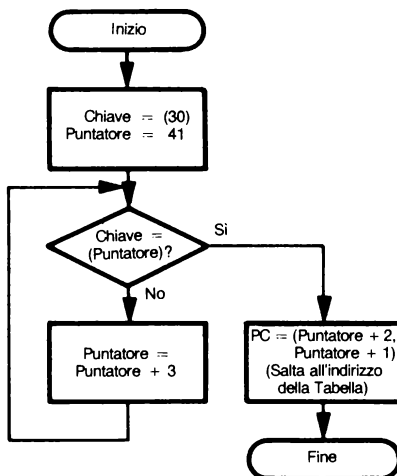
Esistono molti altri algoritmi di classificazione molto variabili in efficienza. Knuth ne descrive alcuni nel libro ricordato precedentemente («The Art of Computer Programming, Volume III: Sorting and Searching»). Kermighan e Plauger descrivono alcuni algoritmi e confrontano la loro efficienza alle Pag. 106 - 111 del loro libro: «The Elements of Programming Style», Mc Graw-Hill, New York, 1971.

Impiego di una Tabella di Salto con Una Chiave

Scopo: Questo programma impiega i contenuti della locazione di memoria 30 come chiave per una tabella di salto iniziante nella locazione 41. Ogni ingresso nella tabella di salto contiene un valore chiave ad 8 bit seguito da un indirizzo a 16 bit (MSB nella seconda parola) per cui il programma trasferirebbe il controllo se la chiave è uguale al valore chiave.

Problema Campione:

(30) = 38
(41) = 32
(42) = 55
(43) = 00
(44) = 35
(45) = 61
(46) = 00
(47) = 38
(48) = 65
(49) = 00
Risultato = (PC) = 0065 poichè questo indirizzo corrisponde al valore chiave 38.

Diagramma di Flusso:

Programma Sorgente:

```
        LDA   30H      ;ACCETTA CHIAVE
        LXI   H,41H    ;PUNTA ALL'INIZIO DELLA TABELLA DI SALTO
SRKEY:  CMP   M        ;CHIAVE È = CHIAVE TABELLA?
        JNX   H
        JZ    FOUND    ;SÌ, SALTA ALL'INDIRIZZO NELLA TABELLA
        INX   H        ;NO, VA AD UN SUCCESSIVO INGRESSO
        INX   H
        JMP   SRKEY
FOUND:  MOV   E,M      ;ACCETTA L'INDIRIZZO DI SALTO DALLA TABELLA
        INX   H
        MOV   D,M
        XCHG
        PCHL           ;INFINE SALTA AD ESSO MUOVENDOLO AL PC
```

Programma Oggetto:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		LDA	30H	3A
01				30
02				00
03		LXI	H,41H	21
04				41
05				00
06	SRKEY:	CMP	M	BE
07		INX	H	23
08		JZ	FOUND	CA
09				10
0A				00
0B		INX	H	23
0C		INX	H	23
0D		JMP	SRKEY	C3
0E				06
0F				00
10	FOUND:	MOV	E,M	5E
11		INX	H	23
12		MOV	D,M	56
13		XCHG		EB
14		PCHL		E9

Se il confronto con una chiave è negativo occorre superare la chiave e l'indirizzo associato prima di eseguire il confronto successivo. Le tre istruzioni INX H fanno questo.

L'istruzione PCHL, che trasferisce H ed L al Contatore di Programma, è molto utile nelle tabelle di salto e nei programmi monitor. Si noti che PCHL è una istruzione di salto poichè posiziona un nuovo valore nel Contatore di Programma. Tutte le istruzioni di Salto e Chiamata impiegano indirizzi fissi.

Non è necessaria l'istruzione di fine poichè PCHL trasferisce il controllo all'indirizzo nella tabella di salto.

Le tabelle di salto sono molto utili in situazioni dove occorre selezionare una o più routine. Tali situazioni si verificano nei comandi di decodifica, nei programmi di selezione test, metodi alternativi di scelta oppure la selezione di una configurazione I/O.

La tabella di salto sostituisce una serie intera di operazioni di salto condizionato. Il programma di ricerca con la tabella di salto potrebbe essere impiegato per la ricerca in alcune tabelle diverse mediante il semplice cambiamento della chiave e dell'indirizzo di partenza.

Anche qui è inopportuno impiegare dati a 16 bit o più lunghi. L'8080 non è dotato di una singola istruzione che possa incrementare l'indirizzo in H ed L di tre e neppure è dotato di una singola istruzione che consideri l'indirizzo di salto a 16 bit e lo estragga dalla tabella. Ognuno di questi compiti richiede alcune istruzioni.

Come si potrebbero ristrutturare le condizioni iniziali per eliminare le ulteriori istruzioni di salto?

PROBLEMI

1) Rimozione di Ingresso dalla Lista

Scopo: Rimuove i contenuti della locazione di memoria 30 da una lista se è presente. La lunghezza della lista è modificata dalla locazione di memoria 41 e la stessa lista inizia dalla locazione 42. Muove gli ingressi sottostanti quelli rimossi eliminando la posizione vuota e riduce la lunghezza della lista di una unità.

Problemi Campione:

a.
(30) = 6B
(41) = 04
(42) = 37
(43) = 61
(44) = 28
(45) = 1D
Risultato = Nessun cambiamento poichè l'ingresso non è contenuto nella lista.

b.
(30) = 6B
(41) = 04
(42) = 37
(43) = 6B
(44) = 28
(45) = 1D
Risultato = (41) = 03
(43) = 2B
(44) = 1D

L'ingresso è rimosso dalla lista e quelli successivi vengono spostati in alto di una posizione. La lunghezza della lista è ridotta di 1.

2) Aggiunta di un Ingresso ad una Lista Ordinata

Scopo: Posiziona i contenuti della locazione di memoria 30 in una lista ordinata se non è già presente. La lunghezza della lista è indicata nella locazione 41, la stessa lista inizia dalla locazione 42 e consiste di numeri binari senza segno in ordine crescente. Posiziona il nuovo ingresso nel punto corretto della lista, aggiusta gli elementi sotto ad esso ed aumenta di 1 la lunghezza della lista.

Problemi Campione:

a.
(30) = 6B
(41) = 04
(42) = 37
(43) = 55
(44) = 7D
(45) = A1
Risultato = (41) = 05
(44) = 6B
(45) = 7D
(46) = A1

b.
(30) = 6B
(41) = 04
(42) = 37
(43) = 55
(44) = 6B
(45) = A1
Risultato = Invariato poichè l'ingresso è già presente in lista.

3) Aggiunta di un Elemento ad una Lista Incatenata

Scopo: Aggiunge l'indirizzo contenuto nelle locazioni di memoria 40 e 41 (MSB in 41) ad una lista incatenata (chained list). Il vecchio indirizzo di partenza si trova nelle locazioni 42 e 43 (MSB in 43). Ogni ingresso alla lista incatenata contiene l'indirizzo del successivo elemento della lista oppure zero se non esiste elemento successivo; gli ingressi sono tutti a 16 bit con i bit più significativi nella seconda parola dell'ingresso stesso. Il nuovo ingresso va in testa alla lista; il suo indirizzo sarà nelle locazioni di memoria 42 e 43 ed esso conterrà l'indirizzo che precedentemente era in quelle locazioni.

Problema Campione:

(40) = 46 } Puntatore alla nuova testa della lista
(41) = 00 }
(42) = 4D } punti di ingresso alla vecchia testa della lista
(43) = 00 }
Risultato = (42) = 46 } nuovo ingresso
(43) = 00 }
(46) = 4D } puntatore alla vecchia testa della lista
(47) = 00 }

4) Classificazione a 16 Bit

Scopo: Classifica un array di numeri binari a 16 bit senza segno, in ordine crescente. La lunghezza dell'array è indicata nella locazione di memoria 40 e l'array stesso inizia nella locazione 41. Ogni numero a 16 bit è immagazzinato con i bit meno significativi nella prima parola.

Problema Campione:

```
      (40)  =  03
      (41)  =  2A
      (42)  =  B5
      (43)  =  60
      (44)  =  3F
      (45)  =  D1
      (46)  =  19
 Risultato  =  (41)  =  D1
               (42)  =  19
               (43)  =  60
               (44)  =  3F
               (45)  =  2A
               (46)  =  B5
```

I numeri sono B52A, 3F60 e 19D1.

5) Impiego di una Tabella di Salto Ordinata

Scopo: Questo programma usa i contenuti della locazione di memoria 30 come indice per una tabella di salto iniziante nella locazione 41. Ogni ingresso nella tabella di salto contiene un indirizzo a 16 bit con gli MSB nella seconda parola a cui il programma trasferirebbe il controllo se l'indice ha un valore appropriato cioè se l'indice è 6, il programma forza il salto all'ingresso della tabella di indirizzo # 6.

Problema Campione:

```
      (30)  =  02
      (41)  =  00
      (42)  =  50
      (43)  =  00
      (44)  =  56
      (45)  =  00
      (46)  =  60
 Risultato  =  (PC)  =  0060 poichè questo è l'ingresso # 2 nella tabella di salto.
```

Capitolo 10
SUBROUTINE

Nessuno degli esempi considerati fin ora è tipicamente un programma tutto da solo. La maggior parte dei programmi reali eseguono una serie di compiti, molti dei quali possono essere lo stesso oppure molti possono essere comuni ad alcuni programmi diversi. È necessario un metodo per formulare questi compiti una volta e rendere questa formulazione convenientemente disponibile sia in parti diverse del programma attuale sia in altri programmi.

BIBLIOTECA DI SUBROUTINE

La risposta sta nella formulazione di compiti come «subroutine». La sequenza di istruzione risultante può essere scritta una sola volta, provata e poi impiegata in modo ripetitivo. Essa può essere la parte di una «biblioteca di subroutine» che fornirà soluzioni documentate ai problemi comuni.

ISTRUZIONI DELLA SUBROUTINE

La maggior parte dei microprocessori è dotata di speciali istruzioni per il trasferimento del controllo alle subroutine e la restituzione del controllo al programma principale. Sarà considerata istruzione speciale che trasferisce il controllo alla subroutine l'istruzione di Chiamata (Call) oppure di Salto alla Subroutine, Salto e Posizionamento Mark oppure Salto e Collegamento. L'istruzione speciale che restituisce il controllo al programma principale è normalmente chiamata Ritorno (Return). Sul microprocessore 8080, l'istruzione di chiamata conserva il vecchio valore del Contatore di Programma nello Stack RAM prima del posizionamento nel Contatore stesso dell'indirizzo di partenza della Subroutine; l'istruzione Ritorno prende il vecchio valore dallo Stack e lo fa rientrare nel Contatore di Programma. L'effetto che si ottiene è il trasferimento del controllo del programma prima alla subroutine e poi indietro al programma principale. Chiaramente la subroutine può essa stessa trasferire il controllo ad un'altra subroutine e così via.

Per essere veramente utile, una subroutine deve essere ragionevolmente generale. Una routine che può eseguire solo un compito specializzato come l'osservazione di una lettera particolare in una stringa di ingresso di lunghezza fissa non sarà molto pratica. Se, d'altra parte, una subroutine può osservare qualunque lettera in stringhe comunque lunghe sarà estremamente utile. I dati e gli indirizzi che la subroutine consente di essere variabili vengono chiamati «parametri». Un ruolo importante nella composizione di una subroutine ha la decisione di quali variabili devono essere considerate parametri.

PASSAGGIO DEI PARAMETRI

Un primo problema è il trasferimento dei parametri alla subroutine; questo processo è chiamato «passaggio» dei parametri. Il metodo più semplice è il posizionamento, da parte del programma principale, dei parametri in registri. Allora la subroutine può semplicemente assumere detti parametri. Naturalmente questa tecnica è limitata dal numero di registri disponibili. I parametri possono, comunque, essere indirizzati come dati. Per esempio una routine di classificazione comincia con l'indirizzo di partenza di un array nei Registri H ed L.

Altri metodi sono necessari quando ci sono molti parametri. Una possibilità è quella di usare lo Stack. Il programma principale può posizionare i parametri nello Stack e la subroutine può recuperarli. I vantaggi di questo metodo sono i seguenti: lo Stack è essenzialmente di dimensioni non limitate ed i dati nello Stack non vanno persi anche se lo Stack è usato ancora. Gli svantaggi sono: poche istruzioni dell'8080 usano lo Stack, inoltre l'istruzione Call immagazzina nel-

lo Stack anche l'indirizzo di ritorno. Un altro metodo è di usare un'area di memoria per i parametri. Il programma principale può posizionare l'indirizzo dell'area dei registri H ed L e la subroutine può poi recuperare i dati come richiesto. Comunque questa procedura è inopportuna se i parametri stessi sono indirizzi.

RIALLOCABILITÀ

Talvolta una subroutine deve avere caratteristiche speciali. Una subroutine è detta «riallocabile» se può essere posizionata ovunque nella memoria. È possibile usare tale subroutine con facilità senza curarsi del posizionamento degli altri programmi o dell'ordinamento della memoria. Un programma strettamente riallocabile può impiegare indirizzi non assoluti; tutti gli indirizzi devono essere all'inizio del programma. Il programma richiede un «caricatore di riallocabilità» per essere posizionato in memoria; il caricatore inizierà il programma dopo gli altri e sommerà l'indirizzo di partenza o «costante di riallocabilità» a tutti gli indirizzi del programma.

SUBROUTINE CON RIENTRO

Una subroutine è dotata di «rientro» se può essere interrotta la sua esecuzione e poi ripristinata dal programma interrupting. La possibilità di rientro è importante per le subroutine standard in un sistema basato sull'interrupt. Altrimenti le routine di servizio interrupt non possono impiegare le subroutine standard senza commettere errori. Le subroutine dei microprocessori vengono facilmente dotate di rientro poichè l'istruzione di Chiamata impiega lo Stack e la procedura è automaticamente rientrante. La sola caratteristica rimanente è che la subroutine deve usare i registri e lo Stack piuttosto che locazioni fisse di memoria per l'immagazzinamento temporaneo. Questo è un po' scomodo ma normalmente può essere fatto, se necessario.

Una subroutine è «recursiva» se chiama sè stessa. Una tale subroutine chiaramente deve essere anche rientrante. Comunque le subroutine recursive sono piuttosto rare nelle applicazioni con microprocessore.

La maggior parte dei programmi consiste di un programma principale e di diverse subroutine. Questo è un vantaggio perchè si possono usare routine già provate e collaudate, per il debugging delle altre subroutine separatamente. Occorre comunque fare molta attenzione ad usare correttamente le varie subroutine ed a ricordare i loro precisi effetti.

DOCUMENTAZIONE DELLA SUBROUTINE

DOCUMENTAZIONE DELLE SUBROUTINE

La lista della subroutine deve fornire sufficienti informazioni cosicchè altri utenti possano utilizzare la subroutine senza dover esaminare la sua struttura interna. Tra le precisazioni necessarie si ricorda:

1) Una descrizione dello scopo della subroutine.
2) Una lista dei parametri.
3) I registri e le locazioni di memoria impiegate.
4) Un caso campione.

Se vengono seguite queste indicazioni la subroutine sarà più facile possibile da usare.

È importante notare che tutti gli esempi seguenti riservano un'area di memoria per lo Stack RAM. Se il monitor del proprio microcomputer stabilisce una certa area, è possibile usare questa al suo posto. Se si desidera provare a fissare la propria area di stack si ricordi di conservare e di immagazzinare il Puntatore dello Stack del monitor in modo da ottenere un corretto ritorno alla fine del programma principale. Per conservare il Puntatore dello Stack del monitor si usi la routine:

```
LXI     H,0     ;ACCETTA IL PUNTATORE DELLO STACK DEL MONITOR
DAD     SP
SHLD    STEMP   ;E CONSERVALO
```

Per prelevare il Puntatore dello Stack del monitor si usi la routine:

```
LHLD    STEMP   ;PRELEVA IL PUNTATORE DELLO STASCK DEL MONITOR
SPHL
```

È stato usato 80 esadecimale come punto di partenza dello Stack. Se necessario, si potrebbe coerentemente sostituire questo indirizzo con uno conveniente per il proprio microcomputer.

ESEMPI

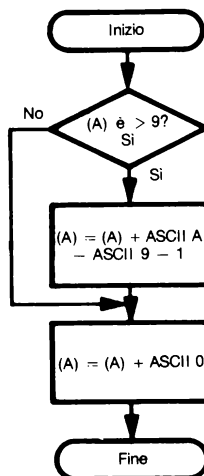
Da Esadecimale ad ASCII

Scopo: Converte i contenuti dell'Accumulatore in un carattere ASCII. Posiziona il risultato nell'Accumulatore.

Problemi Campione:

a. (A) = 0C
Risultato = (A) = 43 'C'

b. (A) = 06
Risultato = (A) = 36 '6'

Diagramma di Flusso:

Programma Sorgente:
Il programma chiamante inizia lo Stack dalla locazione di memoria 80, accetta i dati binari dalla locazione di memoria 40, chiama la subroutine di conversione e immagazzina il risultato nella locazione di memoria 41.

```
        ORG     0
        LXI     SP,80H    ;INIZIA LO STACK DALLA LOCAZIONE DI MEMORIA 80
        LDA     40H       ;ACCETTA DATI
        CALL    ASDEC     ;CONVERTE AD ASCII
        STA     41H       ;IMMAGAZZINA IL RISULTATO
HERE:   JMP     HERE
```

La subroutine converte un digit esadecimale in ASCII:

```
        ORG     20H
ASDEC:  CPI     10        ;IL DATO È 10 O MAGGIORE?
        INC     ASCZ
        ADI     'A'-'9'-1 ;SÌ, AGGIUNGI OFFSET PER LE LETTERE
ASCZ:   ADI     '0'       ;AGGIUNGI OFFSET PER ASCII
        RET
```

Documentazione della subroutine:

```
;
;SUBROUTINE ASDEC
;
;SCOPO: ASDEC CONVERTE UN DIGIT
; ESADECIMALE NELL'ACCUMULATORE IN UN
; CARATTERE ASCII NELL'ACCUMULATORE
;
;CONDIZIONI INIZIALI: DIGIT ESADECIMALE IN A
;
;CONDIZIONI FINALI: CARATTERE ASCII IN A
;
;REGISTRI IMPIEGATI: A
;
;CASO CAMPIONE
; CONDIZIONE INIZIALE: 6 NELL'ACCUMULATORE
; CONDIZIONE FINALE: ASCII 6 (ESADEC. 36)
;   NELL'ACCUMULATORE
;
```

Programma Oggetto:

Programma chiamante:

Indirizzo di Memoria (Esadec.)	Istruzione (Mnemonico)			Contenuti di Memoria (Esadec.)
00		LXI	SP,80H	31
01				80
02				00
03		LDA	40H	3A
04				40
05				00
06		CALL	ASDEC	CD
07				20
08				00
09		STA	41H	32
0A				40
0B				00
0C	HERE:	JMP	HERE	C3
0D				0C
0E				00

Subroutine:

20	ASDEC:	CPI	10	FE
21				0A
22		JNC	ASCZ	D2
23				27
24				00
25		ADI	'A'-'9'-1	C6
26				07
27	ASCZ:	ADI	'0'	C6
28				30
29		RET		C9

L'istruzione LXI SP,80H inizia lo Stack dalla locazione di memoria 80. Si ricordi che lo Stack cresce verso il basso (verso indirizzi più bassi). Normalmente si posizionerà lo Stack all'estremità alta della RAM (cioè l'indirizzo maggiore) in modo che esso non interferirà con l'altro immagazzinamento temporaneo.

L'istruzione di Chiamata posiziona l'indirizzo di partenza della subroutine (0020 esadecimale) nel Contatore di Programma e conserva il vecchio valore del Contatore di Programma (0009 esadecimale) nello Stack. La procedura è:

FASE 1 — Decrementa il Puntatore dello Stack, conserva gli MSB del vecchio valore del Contatore di Programma nello Stack.

FASE 2 — Decrementa il Puntatore dello Stack, conserva gli LSB del vecchio valore del Contatore di Programma nello Stack.

Il risultato in questo caso è:

(7F) = 00
(7E) = 09
(SP) = 7E

Il valore che viene conservato è il valore del contatore di programma dopo che il processore ha prelevato l'intera istruzione di Chiamata dalla memoria. Si noti che l'indirizzo viene immagazzinato proprio come gli altri indirizzi dell'8080 con i bit meno significativi nell'indirizzo più basso.

L'istruzione di Ritorno carica il Contatore di Programma con i contenuti delle due locazioni di memoria del fondo dello Stack. La procedura è

FASE 1 — Muovi gli 8 bit dallo Stack agli LSB del Contatore di Programma. Incrementa il Puntatore dello Stack.
FASE 2 — Muovi gli 8 bit dallo Stack agli MSB del Contatore di Programma. Incrementa il Puntatore dello Stack.

Il risultato in questo caso è:

```
[PC] =(7F) (7E)
     =0009
[SP] = 80
```

Questa subroutine ha un singolo parametro e produce un singolo risultato. L'Accumulatore è la posizione ovvia dove vengono messi entrambi.

Il programma chiamante comprende tre fasi: posizionamento dei dati nell'Accumulatore, chiamata della subroutine e memorizzazione del risultato. Inoltre l'inizializzazione deve assegnare lo Stack ad una appropriata area di memoria.

La subroutine è rientrante poichè non usa locazioni di memoria. La subroutine può essere riposizionata impiegando un diverso statement ORG e poi riassemblando il codice.

Se si presuppone di impiegare lo Stack per i parametri si ricordi che l'istruzione di chiamata posiziona l'indirizzo di ritorno nello Stack. Si può eseguire due volte INX, SP per ottenere l'indirizzo di ritorno ma occorre anche ricordare di aggiustare correttamente il Puntatore dello Stack prima del ritorno. Si può anche muovere il Puntatore dello Stack ai registri H ed L con la sequenza:

```
LXI     H,0
DAD     SP          ;PUNTATORE DELLO STACK AL REGISTRO INDIRIZZO
```

Ora si può usare l'indirizzamento di memoria implicato con H ed L per accedere ai dati nello Stack.

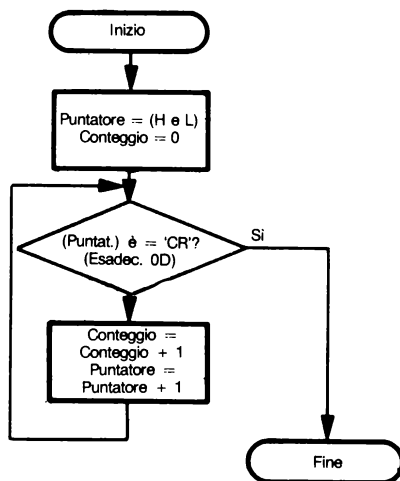
Lunghezza di una Stringa di Caratteri

Scopo: Determina la lunghezza di una stringa di caratteri ASCII. L'indirizzo di inizio della stringa è contenuto nei Registri H ed L.
La fine della stringa è indicata da un carattere di ritorno carrello ('CR' Esadec. 0D).
Posiziona la lunghezza della stringa (escludendo il carattere di ritorno carrello) nell'Accumulatore.

Problemi Campione:

a.
(H ed L) = 43
(43) = 0D
Risultato = (A) = 00

b.
(H ed L) = 43
(43) = 52
(44) = 41
(45) = 54
(46) = 48
(47) = 45
(48) = 52
(49) = 0D
Risultato = (A) = 06

Diagramma di Flusso:

Programma Sorgente:
Il programma chiamante inizia lo Stack dalla locazione di memoria 80, accetta l'indirizzo di partenza dalle locazioni di memoria 40 e 41, chiama la subroutine della lunghezza di stringa ed immagazzina il risultato nella locazione di memoria 42.

```
        ORG     0
        LXI     SP,80H   ;INIZIA LO STACK ALLA LOCAZIONE 80
        LHLD    40H      ;ACCETTA L'INDIRIZZO DI PARTENZA DELLA STRINGA
        CALL    STLEN    ;DETERMINA LA LUNGHEZZA DELLA STRINGA
        STA     42H      ;IMMAGAZZINA LA LUNGHEZZA DELLA STRINGA
HERE:   JMP     HERE
```

La subroutine determina la lunghezza della stringa di caratteri ASCII e posiziona la lunghezza nell'Accumulatore:

```
        ORG     20H
STLEN:  MVI     B,0      ;LUNGHEZZA = 0
        MVI     A,0DH    ;ACCETTA 'CR' PER CONFRONTO
CHKCR:  CMP     M        ;È IL CARATTERE 'CR'?
        JZ      DONE     ;SÌ, FINE DELLA STRINGA
        INR     B        ;NO, AGGIUNGE 1 ALLA LUNGHEZZA
        INX     H
        JMP     CHKCR
DONE:   MOVE    A,B
        RET
```

Documentazione subroutine:

```
;
;SUBROUTINE STLEN
;
;SCOPO: STLEN DETERMINA LA LUNGHEZZA
; DI UNA STRINGA (NUMERO DI CARATTERI
; PRIMA DI UN RITORNO CARRELLO)
;
;CONDIZIONI INIZIALI: INDIRIZZO DI PARTENZA
; DELLA STRINGA IN H ED L
;
;CONDIZIONI FINALI: NUMERO DI CARATTERI IN A
;
;REGISTRI IMPIEGATI: A, B, H, L
;
;CASO CAMPIONE:
; CONDIZIONI INIZIALI: (H ED L) = 43
; (43) = 35, (44) = 44, (45) = 0D
; CONDIZIONE FINALE: (A) = 02
;
```

Programma Oggetto:

Programma chiamante:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		LXI	SP,80H	31
01				80
02				00
03		LHLD	40H	2A
04				40
05				00
06		CALL	STLEN	CD
07				20
08				00
09		STA	42H	32
0A				42
0B				00
0C	HERE:	JMP	HERE	C3
0D				0C
0E				00

Subroutine

20	STLEN:	MVI	B,0	06
21				00
22		MVI	A,0DH	3E
23				0D
24	CHKCR:	CMP	M	BE
25		JZ	DONE	CA
26				2D
27				00
28		INR	B	04
29		INX	H	23
2A		JMP	CHKCR	C3
2B				24
2C				00
2D	DONE:	MOV	A,B	78
2E		RET		C9

Il programma chiamante comprende quattro fasi: inizializzazione del Puntatore dello Stack, posizionamento nei registri H ed L dell'indirizzo di partenza della stringa, chiamata della subroutine ed immagazzinamento del risultato.

La subroutine è rientrante poichè non cambia locazione di memoria.

La subroutine cambia il registro B e l'indirizzo nei registri H ed L come pure l'Accumulatore. Il programmatore deve essere consapevole che i dati immagazzinati nel Registro B e l'indirizzo localizzato in H ed L andranno persi; i registri impiegati sono una parte assolutamente essenziale della documentazione della subroutine.

Un'alternativa alla distruzione dei contenuti del registro nella subroutine è di conservarli nello Stack e poi di rimmagazzinarli prima del ritorno. Questo approccio rende facile all'utente la chiamata della routine ma impiega ulteriore tempo e memoria (nello Stack) nell'esecuzione della routine.

Questa subroutine ha un singolo parametro che è un indirizzo. Il miglior modo per trasmettere questo parametro è di impiegare una coppia di registri e, poichè la Coppia di registri H ed L è certamente la più flessibile, per quanto riguarda la scelta di indirizzamento, essa è la scelta più ovvia.

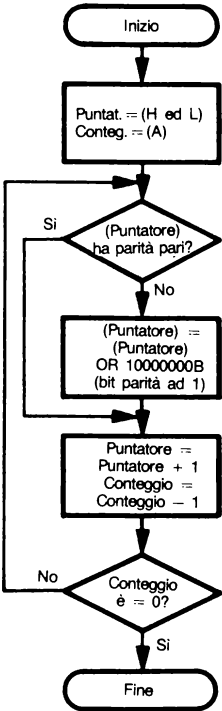
La subroutine contiene una istruzione di salto incondizionato: JMP CHKCR. Alterando le condizioni iniziali precedenti il ciclo della subroutine, è possibile eliminare questo Salto?

Aggiunta di Parità Pari ai Caratteri ASCII

Scopo: Aggiunge parità pari ad una stringa di caratteri ASCII a 7 bit. La lunghezza della stringa è indicata nell'Accumulatore e l'indirizzo di partenza della stringa è nei Registri H ed L. Posiziona la parità pari nel bit più significativo di ogni carattere cioè pone ad 1 il bit più significativo se il numero totale di bit 1 nella parola è pari.

Problema Campione:

(A) = 6
(H ed L) = 40
(40) = 31
(41) = 32
(42) = 33
(43) = 34
(44) = 35
(45) = 36
Risultato = (40) = B1
(41) = B2
(42) = 33
(43) = B4
(44) = 35
(45) = 36

Diagramma di Flusso:

Programma Sorgente:
Il programma chiamante inizia lo Stack dalla locazione di memoria 80, pone l'indirizzo di partenza a 40, accetta la lunghezza della Stringa dalla locazione 30 e chiama la subroutine di parità pari.

```
        ORG     0
        LXI     SP,80H   ;INIZIA LO STACK DALLA LOCAZIONE 80
        LXI     H,40H    ;ACCETTA L'INDIRIZZO DI PARTENZA DELLA STRINGA
        LDA     30H      ;ACCETTA LA LUNGHEZZA DELLA STRINGA
        CALL    EPAR     ;AGGIUNGE PARITÀ PARI ALLA STRINGA
HERE:   JMP     HERE
```

La subroutine aggiunge parità pari alla stringa di caratteri ASCII.

```
        ORG     20H
EPAR:   MOV     B,A
        MVI     C,10000000B ;PONE IL BIT DI PARITÀ AD 1
SETPR:  MOV     A,M         ;ACCETTA UN CARATTERE
        ORA     C           ;BIT PARITÀ = 1
        JPO     CHCNT       ;NON CONSERVA SE ORA PARITÀ È DISPARI
        MOV     M,A         ;RENDE PARITÀ PARI
CHCNT:  INX     H
        DCR     B
        JNZ     SETPR
        RET
```

Documentazione subroutine:

```
;
;SUBROUTINE EPAR
;
;SCOPO: EPAR AGGIUNGE PARITÀ PARI
; AD UNA STRINGA DI CARATTERI ASCII A 7 BIT
;
;CONDIZIONI INIZIALI: INDIRIZZO DI PARTENZA
; DELLA STRINGA IN H ED L, LUNGHEZZA
; DELLA STRINGA IN A
;
;CONDIZIONI FINALI: PARITÀ PARI NELL'MSB
; DI OGNI CARATTERE
;
;REGISTRI IMPIEGATI: A, B, C, H, L
;
;CASO CAMPIONE:
; CONDIZIONE DI PARTENZA: (H ED L) = 40
; (A) = 2; (40) = 32, (41) = 33
; CONDIZIONE FINALE: (40) = B2,
; (41) = 33
;
```

Programma Oggetto:

Programma Chiamante

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		LXI	SP,80H	31
01				80
02				00
03		LXI	H,40H	21
04				40
05				00
06		LDA	30H	3A
07				30
08				00
09		CALL	EPAR	CD
0A				20
0B				00
0C	HERE:	JMP	HERE	C3
0D				0C
0E				00

Subroutine

20	EPAR:	MOV	B,A	47
21		MVI	C,10000000B	0E
22				80
23	SETPR:	MOV	A,M	7E
24		ORA	C	B1
25		JPO	CHCNT	E2
26				29
27				00
28		MOV	M,A	77
29	CHCNT:	INX	H	23
2A		DCR	B	05
2B		JNZ	SETPR	C2
2C				23
2D				00
2E		RET		C9

Il programma chiamante deve posizionare l'indirizzo di partenza della stringa nei Registri H ed L e la lunghezza della stringa in A prima del trasferimento del controllo alla subroutine.

La subroutine cambia i valori nei Registri A, H ed L ed impiega i registri B e C per l'immagazzinamento temporaneo. Essa è rientrante poichè non impiega nessuna locazione di memoria fissa per l'immagazzinamento temporaneo.

Questa subroutine ha due parametri un indirizzo ed un numero. I registri H ed L sono impiegati per trasferire l'indirizzo ed A per trasferire il numero. Nessun risultato esplicito viene ritrasferito al programma chiamante poichè la subroutine influenza solo gli MSB di ogni carattere della stringa.

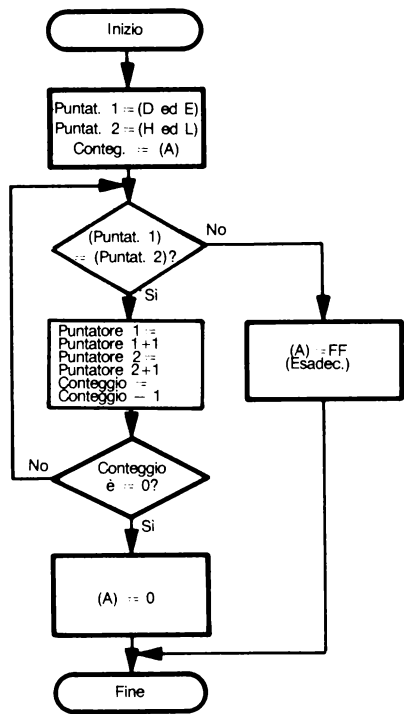
Confronto di Stringhe di Caratteri

Scopo: Questo programma confronta due stringhe di caratteri ASCII per vedere se sono uguali (pattern match). La lunghezza delle stringhe è indicata nell'Accumulatore. L'indirizzo di partenza di una stringa è nei registri H ed L, l'indirizzo di partenza dell'altra nei Registri D ed E. Se le due stringhe sono uguali viene azzerato l'Accumulatore, diversamente si pone l'Accumulatore ad FF esadecimale.

Problemi Campione:

a.
```
      (A)   =  03
 (D ed E)   =  50
 (H ed L)   =  60
     (50)   =  43  C
     (51)   =  41  A
     (52)   =  54  T
     (60)   =  43  C
     (61)   =  41  A
     (62)   =  54  T
Risultato   =  (A)  =  0
```

b.
```
      (A)   =  03
 (D ed E)   =  50
 (H ed L)   =  60
     (50)   =  52  R
     (51)   =  41  A
     (52)   =  54  T
     (60)   =  43  C
     (61)   =  51  A
     (62)   =  54  T
Risultato   =  (A)  =  FF (Esadec.)
```

Diagramma di Flusso:

Programma Sorgente:

Il programma chiamante inizia lo Stack dalla locazione di memoria 80, pone gli indirizzi di partenza delle stringhe a 50 e 60 rispettivamente, accetta la lunghezza di stringa dalla locazione 40, chiama la subroutine per il confronto di strighe e posiziona il risultato nella locazione 41.

```
        ORG     0
        LXI     SP,80H   ;INIZIA LO STACK DALLA LOCAZIONE 80
        LXI     D,60H    ;ACCETTA L'INDIRIZZO DI PARTENZA DELLA STRINGA 1
        LXI     H,50H    ;ACCETTA L'INDIRIZZO DI PARTENZA DELLA STRINGA 2
        LDA     40H      ;ACCETTA LA LUNGHEZZA DI STRINGA
        CALL    PMTCH    ;CONTROLLO PER CONFRONTO
        STA     41H      ;CONSERVA L'INDICATORE DEL CONFRONTO
HERE:   JMP     HERE
```

La subroutine determina se le due stringhe sono uguali.

```
        ORG     20H
PMTCH:  MOV     B,A
CHCAR:  LDAX    D        ;ACCETTA UN CARATTERE DALLA STRINGA 2
        CMP     M        ;COINCIDE CON QUELLA DELLA STRINGA 1?
        JNZ     NOMCH    ;NO, STRINGHE DIVERSE
        INX     D        ;MUOVE PUNTATORI
        INX     H
        DCR     B        ;CONTEGGIO DEI CARATTERI
        JNZ     CHCAR
        SUB     A        ;ACCORDO COMPLETO, (A) = 0
        RET
NOMCH:  MVI     A,0FFH   ;DISACCORDO, (A) = FF
        RET
```

Documentazione subroutine;

```
;
;SUBROUTINE PMTCH
;
;SCOPO: PMTCH DETERMINA SE
; DUE STRINGHE SONO EQUIVALENTI
;
;CONDIZIONI INIZIALI: INDIRIZZI DI PARTENZA
; DELLE STRINGHE IN D ED E, H ED L,
; LUNGHEZZA DELLE STRINGHE NELL'ACCUMULATORE
;
;CONDIZIONI FINALI: O IN A SE LE STRINGHE
; CONCORDANO, ALTRIMENTI FF IN A
;
;REGISTRI IMPIEGATI: A, B, D, E, H, L
;
;CASO CAMPIONE:
; CONDIZIONI INIZIALI: (H ED L) = 50, (D ED E) = 60, (A) = 2
; (50) = 36, (51) = 39
; (60) = 36, (61) = 39
;CONDIZIONI FINALI: (A) = 0 POICHÈ LE STRINGHE CONCORDANO
```

Programma Oggetto:

Programma chiamante:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		LXI	SP,80H	31
01				80
02				00
03		LXI	D,60H	11
04				60
05				00
06		LXI	H,50H	21
07				50
08				00
09		LDA	40H	3A
0A				40
0B				00
0C		CALL	PMTCH	CD
0D				20
0E				00
0F		STA	41H	32
10				41
11				00
12	HERE:	JMP	HERE	C3
13				12
14				00

Subroutine:

20	PMTCH:	MOV	B,A	47
21	CHCAR:	LDAX	D	1A
22		CMP	M	BE
23		JNZ	NOMCH	C2
24				2E
25				00
26		INX	D	13
27		INX	H	23
28		DCR	B	05
29		JNZ	CHCR	C2
2A				21
2B				00
2C		SUB	A	97
2D		RET		C9
2E	NOMCH:	MVI	A,0FFH	3E
2F				FF
30		RET		C9

Questa subroutine, come quelle precedenti, cambia tutti i flag. Si potrebbe assumere in generale, che una subroutine cambi i flag se non è stato specificato diversamente. Se il programma principale ha necessità dei valori precedenti dei flag esso deve conservarli nello Stack prima della chiamata della subroutine. (Questo viene impiegando l'istruzione PUSH PSW).

La subroutine è rientrante e cambia tutti i registri tranne C.

Questa subroutine ha tre parametri — i due indirizzi di partenza e la lunghezza delle stringhe. Questi parametri impiegano cinque dei sette registri di impiego generale (general-purpose).

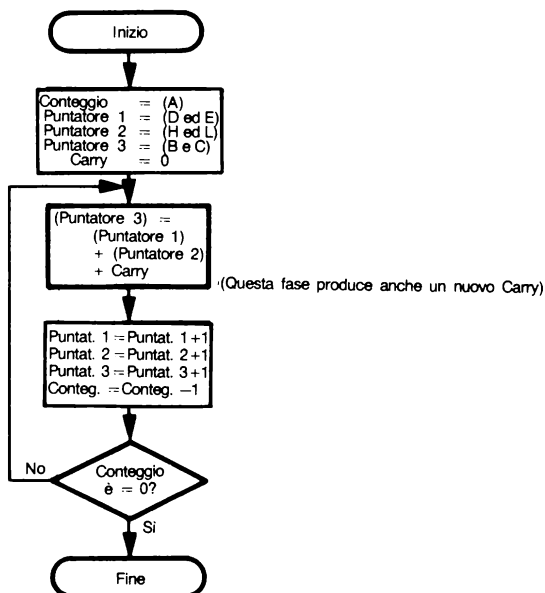
Addizione a Precisione Multipla

Scopo: Somma due numeri binari a parola multipla. La lunghezza dei numeri (in byte) è nell'Accumulatore, gli indirizzi di inizio dei numeri sono nei Registri D ed E ed H ed L e l'indirizzo di partenza del risultato è nei Registri B e C. Tutti i numeri iniziano con i bit meno significativi.

Problema Campione:

(A)	=	04		
(D ed E)	=	51		
(H ed L)	=	61		
(B e C)	=	71		
(51)	=	C3		
(52)	=	A7		
(53)	=	5B		
(54)	=	2F		
(61)	=	B8		
(62)	=	35		
(63)	=	DF		
(64)	=	14		
Risultato	=	(71)	=	7B
		(72)	=	DD
		(73)	=	3A
		(74)	=	44

cioè
```
   2F5BA7C3
 + 14DF35B8
   --------
   443ADD7B
```

Diagramma di Flusso:

Programma Sorgente:

Il programma chiamante inizia lo Stack dalla locazione di memoria 80, pone l'indirizzo di partenza dei diversi numeri a 50, 60 e 70 rispettivamente, accetta la lunghezza dei numeri dalla locazione 40 e chiama la subroutine per l'addizione a precisione multipla.

```
        ORG     0
        LXI     SP,80H  ;INIZIA LO STACK DALLA LOCAZIONE 80
        LXI     H,50H   ;ACCETTA L'INDIRIZZO DI PARTENZA DEL
                        ;PRIMO NUMERO
        LXI     D,60H   ;ACCETTA L'INDIRIZZO DI PARTENZA DEL
                        ;SECONDO NUMERO
        LXI     B,70H   ;ACCETTA L'INDIRIZZO DI PARTENZA DEL
                        ;RISULTATO
        LDA     40H     ;ACCETTA LA LUNGHEZZA DEI NUMERI
        CALL    MPADD   ;ADDIZIONE A PRECISIONE MULTIPLA
HERE:   JMP     HERE
```

La subroutine esegue l'addizione binaria a precisione multipla.

```
        ORG     20H
MPADD:  PUSH    B       ;L'INDIRIZZO DEL RISULTATO ALLO STACK
        MOV     B,A
        ANA     A       ;CARRY INIZIALE = 0
ADDW:   LDAX    D       ;ACCETTA PAROLA DAL PRIMO NUMERO
        ADC     M       ;SOMMA PAROLA DAL SECONDO NUMERO
        XTHL            ;ACCETTA L'INDIRIZZO DEL RISULTATO
        MOV     M,A     ;IMMAGAZZINA PAROLA DEL RISULTATO
        INX     H
        THL             ;CONSERVA L'INDIRIZZO DEL RISULTATO
        INX     D
        INX     H
        DCR     B
        JNZ     ADDW
        POP     B       ;ELIMINA L'INDIRIZZO DEL RISULTATO
                        ;DALLO STACK
        RET
```

Documentazione subroutine:

```
;
;SUBROUTINE MPADD
;
;SCOPO: MPADD SOMMA DUE NUMERI BINARI A PAROLA MULTIPLA
;
;CONDIZIONI INIZIALI: INDIRIZZI DI PARTENZA DEI NUMERI IN D ED E,
; H ED L; INDIRIZZO DI PARTENZA DEL RISULTATO IN B
; E C, LUNGHEZZA DEI NUMERI IN A
;
;REGISTRI IMPIEGATI: TUTTI
;
;CASO CAMPIONE:
; CONDIZIONI DI PARTENZA: (H ed L) = 50, (D ed E) =
; 60, (B e C) = 70, (A) = 2, (50) = C3, (51) = A7; (60) = B8,
; (61) = 35
; CONDIZIONI FINALI: (70) = 7B, (71) = DD
```

Programma Oggetto:
Programma chiamante:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		LXI	SP,80H	31
01				80
02				00
03		LXI	H,50H	21
04				50
05				00
06		LXI	D,60H	11
07				60
08				00
09		LXI	B,70H	01
0A				70
0B				00
0C		LDA	40H	3A
0D				40
0E				00
0F		CALL	MPADD	CD
10				20
11				00
12	HERE:	JMP	HERE	C3
13				12
14				00

Subroutine

20	MPADD:	PUSH	B	C5
21		MOV	B,A	47
22		ANA	A	A7
23	ADDW:	LDAX	D	1A
24		ADC	M	8E
25		XTHL		E3
26		MOV	M,A	77
27		INX	H	23
28		XTHL		E3
29		INX	D	13
2A		INX	H	23
2B		DCR	B	05
2C		JNZ	ADDW	C2
2D				23
2E				00
2F		POP	B	C1
30		RET		C9

L'istruzione XTHL scambia le due locazioni alla sommità dello Stack con i contenuti dei registri H ed L. Questa singola istruzione accetta l'indirizzo per il risultato dallo Stack e conserva l'indirizzo dei dati nello Stack. Essa consente di usare la sommità dello Stack come un'ulteriore coppia di registri.

L'istruzione POP B alla fine del programma azzera i dati provenienti dallo Stack in modo che l'istruzione di Ritorno operi correttamente. Occorre fare attenzione ad azzerare correttamente lo Stack poichè l'istruzione di ritorno usa come sua destinazione l'ingresso alla sommità dello Stack.

Questa subroutine ha quattro parametri — tre indirizzi e la lunghezza dei numeri. Tutti i registri sono impiegati per trasmettere parametri.

La subroutine richiede alcuni registri per il suo impiego e deve perciò utilizzare alcuni registri di immagazzinamento temporaneo. Lo Stack non solo è comodo da usare per questo impiego ma consente anche al programma di essere rientrante.

PROBLEMI

Si noti che è necessario scrivere sia il programma chiamante per il problema campione sia la subroutine correttamente documentata.

1) Da ASCII ad Esadecimale

Scopo: Converte i contenuti dell'Accumulatore dalla rappresentazione ASCII di un digit esadecimale allo stesso digit effettivo. Posiziona il risultato nell'Accumulatore.

Problemi Campione:

a. (A) = 43 'C'
Risultato = (A) = 0C

b. (A) = 36 '6'
Risultato = (A) = 06

2) Lunghezza di Un Messaggio di Telescrivente

Scopo: Determina la lunghezza di un messaggio di telescrivente codificato ASCII. L'indirizzo di partenza della stringa di caratteri, supporto del messaggio, è nei registri H ed L. Il messaggio inizia con un carattere ASCII STX (esadec. 02) e termina con ETX (esadec. 03). Posiziona la lunghezza del messaggio (cioè il numero di caratteri tra STX ed ETX) nell'Accumulatore.

Problema Campione:

(H ed L) = 41
(41) = 49
(42) = 02 STX
(43) = 47 G
(44) = 4F 0
(45) = 03 ETX
Risultato = (A) = 02

3) Controllo di Parità Pari di Caratteri ASCII

Scopo: Controlla la parità di una stringa di caratteri ASCII. La lunghezza della stringa è nell'Accumulatore e l'indirizzo d'inizio della stringa è nei Registri H ed L. Se la parità di tutti i caratteri della stringa è pari, azzera l'Accumulatore, diversamente pone l'Accumulatore ad FF esadec. (tutti uni).

Problemi Campione:

a. (A) = 03
(H ed L) = 42
(42) = B1
(43) = B2
(44) = 33
Risultato = (A) = 00 poichè tutti i caratteri hanno parità pari.

b. (A) = 03
(H ed L) = 42
(42) = B1
(43) = B6
(44) = 33
Risultato = (A) = FF (esadec.) poichè il carattere della locazione di memoria 43 non ha parità pari.

4) Confronto di Stringhe

Scopo: Confronta due stringhe di caratteri ASCII per vedere qual è la maggiore (cioè quale segue l'altra in ordine 'alfabetico'). La lunghezza delle stringhe è nell'Accumulatore; l'indirizzo di inizio della stringa 1 è nei Registri H ed L e l'indirizzo di inizio della stringa 2 è nei Registri D ed E. Se la stringa 1 è maggiore o uguale alla 2 azzera l'Accumulatore, diversamente pone l'Accumulatore ad FF esadec. (tutti uni).

Problemi Campione:

a. (A) = 03
(D ed E) = 60
(H ed L) = 50
(50) = 43 C
(51) = 41 A
(52) = 54 T
(60) = 42 B
(61) = 41 A
(62) = 54 T
Risultato = (A) = 00 poichè CAT è 'maggiore' di BAT.

b. (A) = 03
(D ed E) = 60
(H ed L) = 50
(50) = 44 D
(51) = 4F O
(52) = 47 G
(60) = 44 D
(61) = 4F O
(62) = 47 G
Risultato = (A) = 00 poichè le due stringhe sono uguali.

c. (A) = 03
(D ed E) = 60
(H ed L) = 50
(50) = 43 C
(51) = 41 A
(52) = 54 T
(60) = 43 C
(61) = 55 U
(62) = 54 T
Risultato = (A) = FF (esadec.) Poichè CUT è 'maggiore' di CAT.

5) Sottrazione Decimale

Scopo: Sottrae due numeri decimali (BCD) a parola multipla. La lunghezza dei numeri (in byte) è nell'Accumulatore, gli indirizzi di inizio dei numeri sono nei registri D ed E e H ed L (sottrae quello con indirizzo di inizio in H ed L). L'indirizzo di inizio del risultato è nei Registri B e C. Tutti i numeri iniziano con i digit meno significativi. Al Ritorno il segno del risultato è nell'Accumulatore — zero se il risultato è positivo, FF (esadec.) se è negativo.

Problema Campione:

```
       (A)  =  04
  (H ed L)  =  50
  (D ed E)  =  60
   (B e C)  =  70
      (50)  =  85
      (51)  =  19
      (52)  =  70
      (53)  =  36
      (60)  =  59
      (61)  =  34
      (62)  =  66
      (63)  =  12
 Risultato  =  (A)  = 00 (positivo)
               (70) = 26
               (71) = 85
               (72) = 03
               (73) = 24
      cioè     36701985
            -  12663459
            +  24038526
```

Capitolo 11
INGRESSO/USCITA

Esistono due problemi fondamentali nel progetto di sistemi di ingresso/uscita: il primo riguarda il collegamento delle periferiche al computer ed il trasferimento dei dati, dei segnali di stato e controllo; il secondo problema è come indirizzare i dispositivi di I/O in modo che la CPU possa selezionarne uno particolare per un trasferimento di dati. Chiaramente il primo problema è più complesso e più interessante. Si discuterà perciò del collegamento delle periferiche e si lascerà l'indirizzamento ad un testo più orientato verso l'hardware.

In teoria il trasferimento dei dati a oppure da un dispositivo di I/O non è molto diverso dal trasferimento di dati alla oppure dalla memoria. Infatti si può considerare la memoria proprio come un dispositivo di I/O. Comunque la memoria è speciale per le ragioni seguenti:

I/O E MEMORIA

1) Essa opera quasi alla stessa velocità del processore.
2) Essa impiega lo stesso tipo di segnali binari in tensione della CPU. Il solo dispositivo normalmente richiesto per collegare la memoria e la CPU sono i driver ed i receiver.
3) Essa non richiede un formato speciale e nessun segnale di controllo oltre agli impulsi di LETTURA/SCRITTURA.
4) Rivela automaticamente i dati che le sono inviati.
5) La sua lunghezza base di parola è la stessa di quella del computer.

Molti dispositivi di I/O non hanno tali caratteristiche convenienti. Essi possono operare a velocità molto più basse del microprocessore; per esempio una telescrivente può trasferire solo 10 caratteri al secondo mentre un processore lento può trasferire 10.000 caratteri al secondo — Anche la gamma di velocità è molto larga — i sensori possono fornire una lettura al minuto mentre i CRT oppure i floppy disk possono trasferire 250.000 bit al secondo. Inoltre i dispositivi di I/O possono richiedere segnali continui (per esempio, motori o termometri), correnti piuttosto che tensioni (per esempio una telescrivente), oppure segnali in tensione molto diversi dai segnali impiegati dal microprocessore (per esempio display a scarica di gas). I dispositivi di I/O possono richiedere anche formati speciali, protocolli o segnali di controllo. La loro lunghezza base di parola può essere molto più corta o molto più lunga della lunghezza di parola del computer. Tutte queste variazioni comportano che il progetto di sistemi di I/O è difficile ed ha pochi principi generali. Ogni periferica va considerata da sola e presenta un proprio problema speciale di collegamento.

CATEGORIE DI I/O

È possibile fornire una descrizione generale dei dispositivi e dei metodi di realizzazione delle interfacce. Si può in prima approssimazione dividere i dispositivi in tre categorie a seconda delle loro velocità dei dati:

1) Dispositivi lenti che cambiano stato non più di una volta al secondo ed i cui cambiamenti di stato tipicamente hanno una durata dell'ordine del millisecondo o maggiore. Tali dispositivi comprendono display luminosi, interruttori, relé e molti sensori ed attuatori meccanici.

2) Dispositivi a media-velocità che hanno velocità di trasferimento dati da 1 a 10.000 bit al secondo. Tali dispositivi comprendono tastiere, stampanti, lettori di schede e nastro di carta e

perforatori, driver di cassette, linee di comunicazione ordinaria e molti sistemi analoghi per l'acquisizione dati.

3) Dispositivi ad alta velocità che hanno velocità di trasferimento dati superiore a 10.000 bit al secondo. Tali dispositivi comprendono nastri e dischi magnetici, stampanti ad elevata velocità di linea, linee di comunicazione ad alta velocità e video display.

REALIZZAZIONE DELL'INTERFACCIA CON DISPOSITIVI LENTI

La realizzazione dell'interfaccia con dispositivi lenti è relativamente semplice. Sono necessari pochi segnali di controllo eccetto che per il multiplexing cioè il maneggiamento di diversi dispositivi da una sola porta come mostrato nelle Figure da 11-1 ad 11-4. I dati di ingresso da dispositivi lenti non richiedono un latch a causa delle lunghe costanti di tempo implicate. I dati di uscita devono, naturalmente, avere i latch. Gli unici problemi che si hanno mentre i dati stanno entrando sono le trasmissioni nell'istante di campionamento dei dati di ingresso. I circuiti hardware oppure le routine di ritardo software possono attenuare i periodi di transizione.

Una singola porta può maneggiare diversi dispositivi lenti. La Figura 11-1 mostra un demultiplexer che dirige automaticamente i dati al successivo dispositivo consecutivo mediante il conteggio delle operazioni di uscita. La Figura 11-2 mostra una porta di controllo che fornisce le linee selezionate ad un demultiplexer. L'ordine richiesto qui non è consecutivo ma una istruzione di uscita è necessaria per cambiare lo stato della porta di controllo. Il demultiplexer di uscita è comunemente impiegato per pilotare diversi display dalla stessa porta di uscita. Le Figure 11-3 ed 11-4 mostrano le stesse alternative per un multiplexer di ingresso.

Si notino le differenze tra ingresso ed uscita con dispositivi lenti:

1) I dati di ingresso non richiedono il latch poichè i dispositivi di ingresso forniscono i dati per una lunghezza di tempo enorme mediante gli standard del computer. I dati di uscita devono avere il latch poichè i dispositivi di uscita non risponderanno a dati che sono presenti solo per alcuni cicli di clock della CPU.

2) Le transizioni d'ingresso sono il maggior problema a causa della loro lunghezza; le transizioni d'uscita non costituiscono problemi a causa della lenta reazione del dispositivo d'uscita rispetto al ciclo di clock della CPU.

3) Le maggiori costrizioni sull'ingresso sono il tempo di reazione e quello di assenza di risposta; le maggiori costrizioni sull'uscita sono il tempo di risposta e l'osservabilità.

I dispositivi a media velocità devono essere in qualche modo sincronizzati al clock del processore. La CPU non può trattare semplicemente questi dispositivi come se questi conservassero per sempre i loro dati o potesse ricevere dati in qualunque momento. Inoltre la CPU deve avere un modo per scoprire quando in un dispositivo è presente un nuovo dato di ingresso oppure quando un dispositivo è pronto per ricevere dati d'uscita. Essa deve inoltre avere un modo per informare un dispositivo d'uscita che un nuovo dato d'uscita è pronto.

HANDSHAKE

La procedura standard senza clock è l'handshake. Qui il mittente trasferisce i dati ed indica la presenza di dati al ricevitore; il ricevitore legge i dati e completa l'handshake riconoscendo la ricezione. Il ricevitore può controllare la situazione richiedendo i dati dall'inizio oppure indicando la sua indisposizione ad accettare dati; il mittente poi invia i dati e completa l'handshake inviando un indicatore della presenza di dati. In entrambi i casi il mittente conosce se il trasferimento è stato eseguito con successo ed il ricevitore sa quando è presente un nuovo dato.

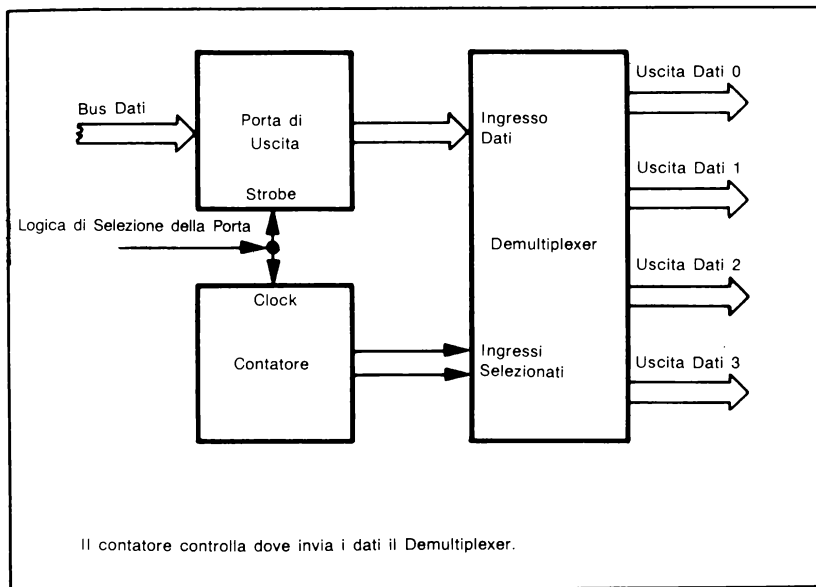



Figura 11-1
Un Demultiplexer Di Uscita Controllato Da Un Contatore

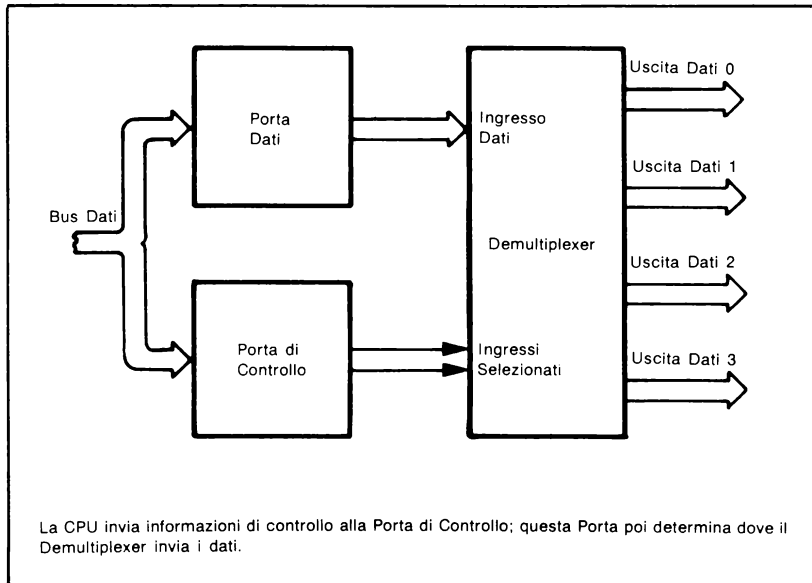



Figura 11-2
Un Demultiplexer Di Uscita Controllato Da Una Porta

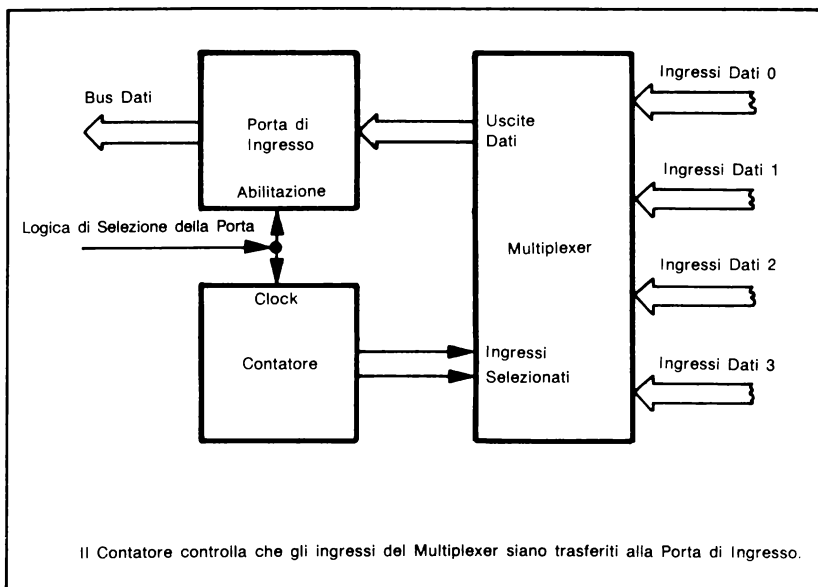



Figura 11-3
Un Multiplexer Controllato Da Un Contatore

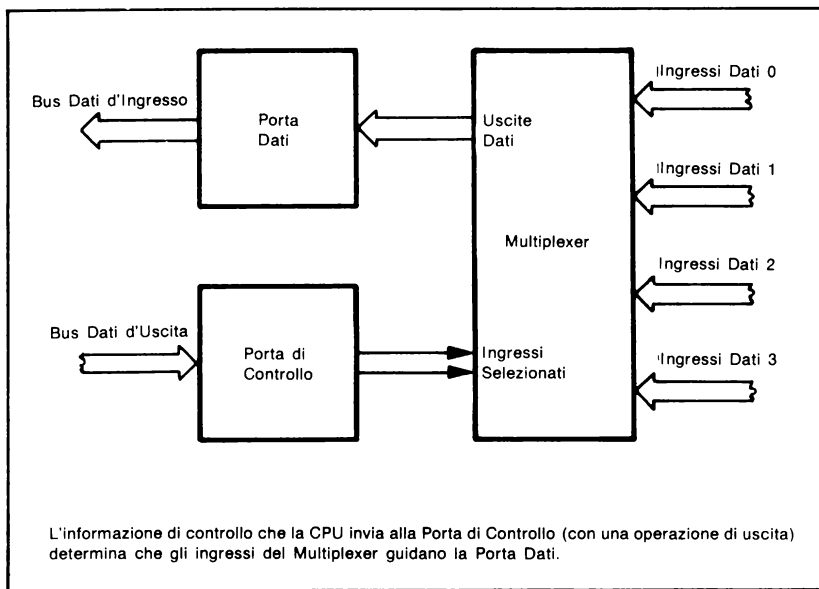



Figura 11-4
Un Multiplexer Di Ingresso Controllato Da Una Porta

Le figure 11-5 ed 11-6 mostrano operazioni tipiche d'ingresso e d'uscita impieganti il metodo handshake. La procedura secondo la quale la CPU esegue un'operazione d'ingresso per controllare la prontezza di una periferica prima del trasferimento dati è chiamata «polling» (sondaggio o registrazione).

Chiaramente il polling può occupare una grande quantità del tempo del processore se ci sono molti dispositivi di I/O. Esistono diversi modi per fornire i segnali di handshake. Tra questi si ricordano:

- Linee di I/O dedicate separatamente. Il processore può maneggiarle attraverso il suo normale sistema di I/O oppure attraverso speciali linee di interrupt. Il processore 8085 è dotato di una speciale linea di ingresso seriale (SIP) e di una speciale linea di uscita seriale (SOD); il processore 8080 non possiede queste linee.

- Schemi speciali su linee di I/O. Questi possono essere bit di singola partenza oppure arresto oppure interi caratteri o gruppi di caratteri. Lo schema deve essere facilmente distinguibile dal rumore di fondo o dallo stato ordinario delle linee.

STROBE

Spesso si chiama una linea di I/O separata che indica la presenza di dati o la necessità di un'operazione indicandola con «strobe». Uno strobe può, per esempio, posizionare dati in un latch o prelevarli da un buffer.

Molte periferiche trasferiscono dati ad intervalli regolari, cioè in modo sincrono. Qui il solo problema si ha alla partenza del processo, cioè l'allineamento del primo ingresso e l'individuazione della prima uscita. In alcuni casi il primo trasferimento procede in modo asincrono; in altri casi la periferica fornisce un segnale di clock che il processore può esaminare per scopi di temporizzazione (timing).

RIDUZIONE DEGLI ERRORI DI TRASMISSIONE

Un problema importante con i dispositivi a media velocità sono gli errori nella trasmissione. Diversi metodi possono diminuire la probabilità di tali errori; tra questi si notano:

- Campionamento dei dati di ingresso al centro dell'intervallo di trasmissione per evitare effetti di transizione.

- Campionamento di ogni ingresso eseguito diverse volte, impiegando la logica maggioritaria, per esempio se ci sono cinque linee di ingresso e tre o più di queste sono 'on' allora il segnale è assunto essere 'on'.

- Generazione e controllo della parità, cioè un bit ulteriore che rende il numero di bit 1 nel dato corretto pari o dispari.

- Impiego di codici a rivelazione e correzione di errore come checksum, LRC (controllo a ridondanza longitudinale) e CRC (Controllo a ridondanza ciclica).

ACCESSO DIRETTO IN MEMORIA

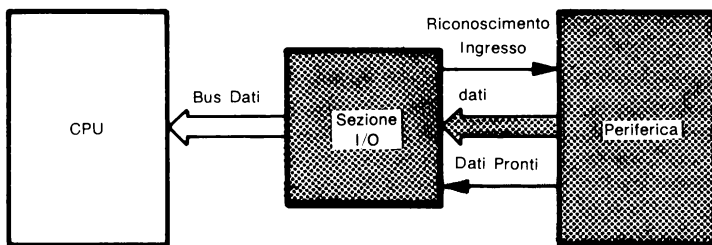
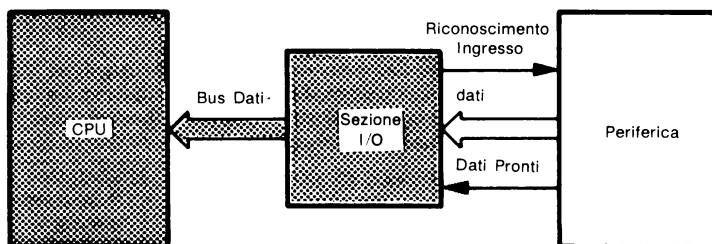
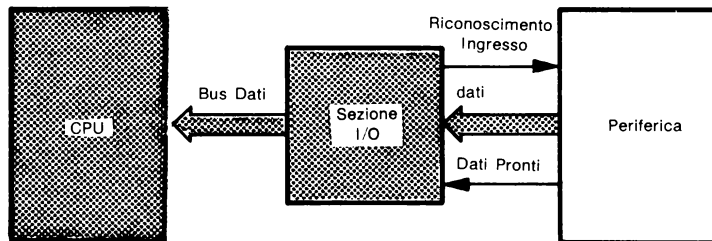
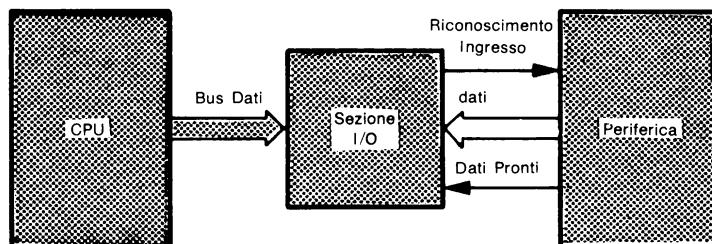
I dispositivi ad alta velocità che trasferiscono più di 10.000 bit al secondo richiedono metodi speciali. La tecnica più comune è quella di costruire un controllore esterno per scopi speciali che trasferisce dati direttamente tra la memoria ed il dispositivo di I/O. Questo processo è chiamato accesso diretto in memoria (DMA). Il controllore DMA deve forzare la CPU a porre off i bus, fornire indirizzi e segnali di controllo alla memoria e trasferire i dati. Tale controllore sarà abbastanza complesso e tipicamente sarà formato da 50 a 100 chip sebbene siano attualmente disponibili dispositivi LSI. Per esempio il Controllore ad Accesso Diretto in Memoria 8257 per i microcomputer basati sull'8080 è descritto in <u>An Introduction To Microcomputers: Volume II. Some Real, Products.</u> La CPU deve inizialmente caricare l'Indirizzo ed i Contatori dei Dati nel controllore in modo che il controllore sappia dove e quanti dati trasferire.

a) La periferica fornisce dati ed il segnale di Dati Ready alla sezione I/O di un computer.

b) La CPU legge il segnale Dati Ready dalla Sezione I/O (questa connessione può essere hardware, per esempio interrupt).

c) La CPU legge i dati dalla sezione I/O.

d) La CPU invia il segnale di Riconoscimento Ingresso alla Sezione I/O che poi fornisce il segnale di Riconoscimento Ingresso alla Periferica (questo può essere una connessione hardware).

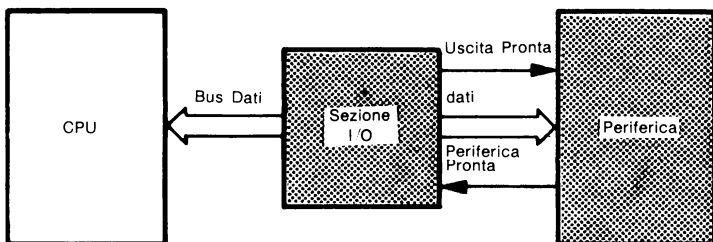
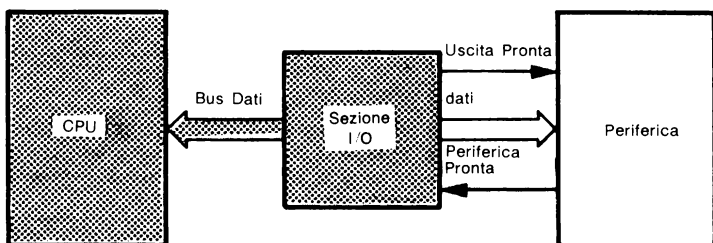
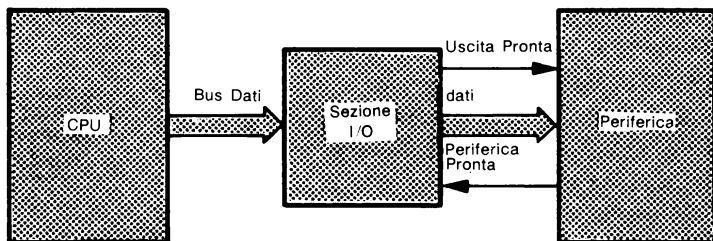
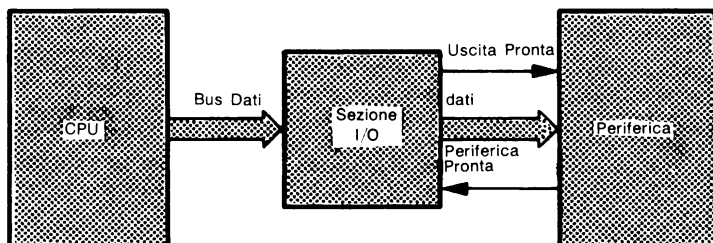
Figura 11-5. Un Handshake Di Ingresso

a) La Periferica fornisce il segnale Periferica Pronta alla sezione I/O del computer.

b) La CPU legge il segnale di Periferica Pronta dalla sezione I/O (questa connessione può essere hardware, per esempio interrupt).

c) La CPU invia i dati alla Periferica.

d) La CPU invia il segnale di Uscita Pronta alla Periferica (questo può essere una connessione hardware).

Figura 11-6. Un Handshake Di Uscita

INTERVALLI DI TIMING (RITARDI)

USO DEGLI INTERVALLI DI TIMING

Un problema che si affronterà attraverso la discussione di ingresso/uscita è la generazione di intervalli di timing (temporizzazione) di una lunghezza specifica. Tali intervalli sono necessari per interruttori meccanici (cioè per appianare le loro transizioni irregolari), per fornire impulsi con lunghezze e frequenze specifiche per display e per fornire timing per dispositivi che inviano o ricevono dati regolarmente (per esempio una telescrivente che invia o riceve un bit ogni 9,1 ms).

METODI PER PRODURRE INTERVALLI DI TIMING

Si possono produrre intervalli di timing in diversi modi:

1) In hardware con un metodo 'one-shot' (multibratori monostabili). Questi dispositivi producono un singolo impulso di durata fissata in risposta ad un ingresso trigger.
2) Con una combinazione di hardware e software con un timer flessibile e programmabile, come il Timer Programmabile 8253 per microcomputer basati sull'8080 come descritto in Un Introduction To Microcomputers, Volume II – Some Real Products. Il timer 8253 può fornire intervalli di timing di varie lunghezze con una varietà di condizioni di inizio e fine.
3) In software con routine di ritardo. Queste routine impiegano il processore come contatore. Questo impiego è possibile se il processore ha un clock di riferimento stabile, ma è chiaramente un sottoutilizzo del processore. Comunque le routine di ritardo non richiedono hardware addizionale e spesso usano del tempo del processore che diversamente sarebbe sprecato.

SCELTA DI UN METODO DI TIMING

La scelta tra questi tre metodi dipende dalle applicazioni. Il metodo software non è dispendioso ma può sovraccaricare il processore. I timer programmabili sono relativamente dispendiosi ma sono facili da collegare e possono essere in grado di maneggiare molti compiti complessi di timing. L'uso del metodo one-shot è normalmente evitato dai migliori progettisti digitali.

ROUTINE DI RITARDO

RITARDO SOFTWARE DI BASE

Una semplice routine di ritardo lavora come segue:

FASE 1 – Carica un registro con un valore specifico.
FASE 2 – Decrementa il registro.
FASE 3 – Se il risultato della FASE 2 non è zero, ripete la FASE 2.

Questa routine è strettamente impiegata per scopi di timing, essa non può eseguire altre funzioni. La quantità di tempo impiegata dipende dal tempo di esecuzione delle varie istruzioni e dal valore specifico caricato nel registro. La massima lunghezza del ritardo è limitata dalle dimensioni del registro; comunque l'intera routine può essere posizionata all'interno di una routine simile che impiega un altro registro e così via.

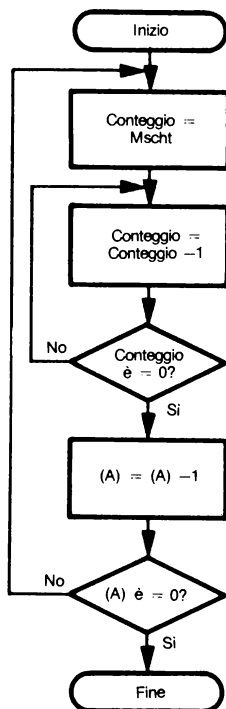
ROUTINE DI RITARDO TRASPARENTE

Il seguente esempio impiega il Registro C e l'Accumulatore per fornire ritardi lunghi 255ms. La scelta dei registri è arbitraria. È possibile infatti trovare più conveniente l'uso di una coppia di registri (per esempio B e C). Una istruzione PUSH B all'inizio della routine di ritardo e l'istruzione POP B alla fine formeranno una routine che non influenza nessun registro. Si noti che le istruzioni PUSH e POP devono essere comprese nel bilancio del tempo.

ESEMPI

Programma di Ritardo

Scopo: Questo programma fornisce un tempo di ritardo di 1 ms ai contenuti dell'Accumulatore.

Diagramma di Flusso:

Il valore di MSCNT dipende dalla velocità della CPU e dal ciclo di memoria.

Programma Sorgente:

```
DELAY:  MVI    B,MSCNT ;ACCETTA CONTEGGIO PER IL RITARDO DI 1 MS
DLY1:   DCR    B       ;CONTEGGIO = CONTEGGIO −1
        JNZ    DLY1    ;CONTINUA FINO A CHE CONTEGGIO = 0
        DCR    A       ;NUMERO DI MS = NUMERO DI MS −1
        JNZ    DELAY   ;CONTINUA FINO A CHE NUMERO DI MS = 0
        RET
```

Programma Oggetto: (iniziante dalla locazione 30)

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
30	DELAY:	MVI	B,MSCNT	06
31				MSCNT
32		DCR	B	05
33		JNZ	DLY1	C2
34				32
35				00
36		DCR	A	3D
37		JNZ	DELAY	C2
38				30
39				00
3A		RET		C9

Bilancio del Tempo:

Istruzione		Numero di volte eseguita
MVI	B,MSCNT	(A)
DCR	B	(A) x MSCNT
JNZ	DLY1	(A) x MSCNT
DCR	A	(A)
JNZ	DELAY	(A)
RET		

Il tempo totale impiegato dovrebbe essere (A) x 1 ms. Se la memoria operasse alla massima velocità, le istruzioni impiegherebbero il seguente numero di cicli di clock:

MVI	B	7
DCR		5
JNZ		10

Trascurando le istruzioni CALL e RET (che vengono eseguite una sola volta): il programma impiega:

$$(A) \times (7 + 15 \times MSCNT + 15)$$

cicli di clock. Così per ottenere il ritardo di 1 ms si ha:

$$22 + (15 \times MSCNT) = N_C$$

dove N_C è il numero di cicli di clock per millisecondo. Alla velocità di clock standard per l'8080 di 2MHz, $N_C = 2000$ allora:

$$15 \times MSCNT = 1978$$

MSCNT = 132 (Esadec. 84) per una velocità di clock di 2MHz per l'8080.

COSTANTE DEL CICLO DI RITARDO PER L'8080A

Il timing dell'8085 è diverso poichè l'istruzione DCR impiega 4 cicli di clock su questo processore. Inoltre JNZ richiede soltanto 7 cicli di clock quando il salto viene eseguito.

COSTANTE DEL CICLO DI RITARDO PER L'8085

Cosicchè per produrre un ritardo di 1ms sull'8085, l'equazione è:

$$7 + 14 \times MSCNT - 3 + 11 = Nc$$

ovvero:

$$(14 \times MSCNT) + 15 = Nc$$

L'addendo −3 è originato dal fatto che l'ultima istruzione JNZ senza successo all'interno del ciclo impiega 7 piuttosto che 10 cicli. Alla velocità di clock standard di 3MHz per l'8085 è Nc = 3000 cosicchè:

$$14 \times MSCNT = 2985$$

MSCNT = 213 (Esadec. D5) per una velocità di clock di 3 MHz per l'8085

Un Pulsante (ovvero SPST Interruttore Istantaneo)

Scopo: Realizzare l'interfaccia tra un interruttore a pulsante (ovvero un interruttore istantaneo unipolare ad una sola via: SPST) ed un microprocessore 8080. Il pulsante è un interruttore meccanico che fornisce la chiusura di un singolo contatto (cioè uno '0' logico) mentre viene premuto.

Schema del Circuito:
La Figura 11-7 mostra la circuiteria richiesta per realizzare l'interfaccia con il pulsante. Questa realizzazione impiega un bit di una porta d'ingresso 8212 che funziona come buffer; non è richiesto nessun latch poichè la chiusura del pulsante è presente per un tempo molto lungo entro gli standard della CPU. (In questo libro non verrà descritto in dettaglio l'uso della porta 8212. È possibile trovare una descrizione completa di tale dispositivo nel Volume II di An Introduction To Microcomputers).

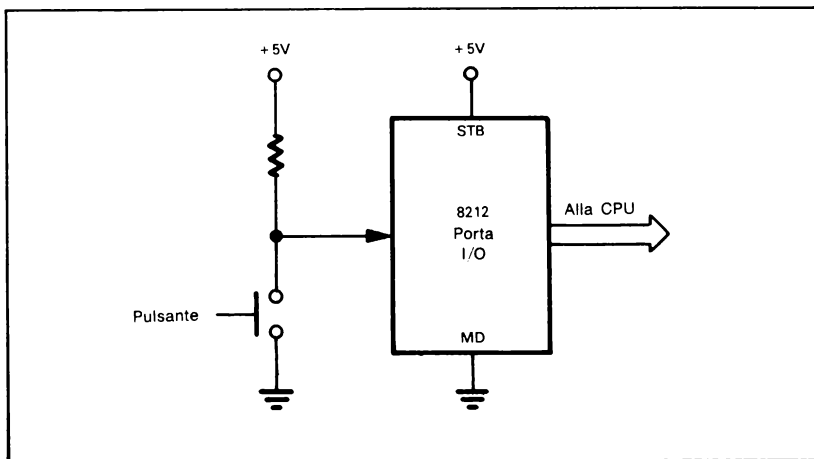



Figura 11-7. Un Circuito A Pulsante

Esempi di Programmazione:
Saranno sviluppati due compiti che fanno uso di questo circuito. Questi sono:

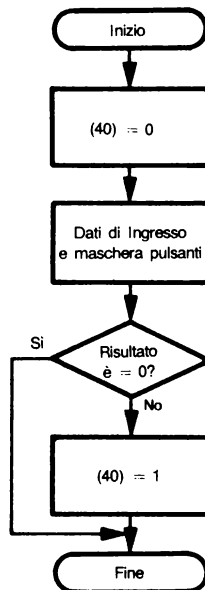
1) Assegnazione di una locazione di memoria in base alla posizione dell'interruttore.
2) Conteggio del numero di volte che l'interruttore viene chiuso.

Compito 1: Pone la locazione di memoria 40 ad 1 se il pulsante è aperto ('1'), a 0 se è chiuso ('0').

Problemi Campione:

a. Pulsante aperto (cioè non premuto)
   Risultato = (40) = 01

b. Pulsante chiuso (cioè premuto)
   Risultato = (40) = 00

Diagramma di Flusso:

Programma Sorgente:

```
        LXI     H,40H
        MVI     M,0       ;ASSEGNAZIONE = 0
        IN      PORT      ;LEGGE POSIZIONE PULSANTE
        ANI     MASK      ;IL PULSANTE È CHIUSO (0)?
        JZ      DONE      ;SÌ, SALTA A DONE
        INR     M         ;NO, ASSEGNAZIONE = 1
DONE:   JMP     DONE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		LXI	H,40H	21
01				40
02				00
03		MVI	M,0	36
04				00
05		IN	PORT	DB
06				PORT
07		ANI	MASK	E6
08				MASK
09		JZ	DONE	CA
0A				0D
0B				00
0C		INR	M	34
0D	DONE:	JMP	DONE	C3
0E				0D
0F				00

Il numero della porta dipende da quale porta è connessa al pulsante. La MASK dipende da quale bit della porta il pulsante è connesso. La MASK (maschera) ha un bit '1' nella posizione del pulsante e zero altrove.

Posizione Pulsante (Numero Bit)	Maschera	
	Binario	Esadecimale
0	00000001	01
1	00000010	02
2	00000100	04
3	00001000	08
4	00010000	10
5	00100000	20
6	01000000	40
7	10000000	80

Se il pulsante è connesso al bit 0 oppure al bit 7 della porta d'ingresso, il programma può impiegare un'istruzione di Scorrimento per porre ad 1 il Carry e quindi determinare lo stato del pulsante. Per esempio:

```
Bit 7
IN      PORT    ;LEGGI POSIZIONE PULSANTE
RAL             ;IL PULSANTE È CHIUSO (0)?
JNC     DONE    ;SÌ, SALTA A DONE

Bit 0
IN      PORT    ;LEGGI POSIZIONE PULSANTE
RAR             ;IL PULSANTE È CHIUSO (0)?
JNC     DONE    ;SÌ, SALTA A DONE
```

Se il pulsante è connesso ai bit 6 o 7 della porta d'ingresso il programma può impiegare il bit Segno per determinare lo stato del pulsante. Per esempio:

```
Bit 7
IN      PORT    ;LEGGI POSIZIONE PULSANTE
ORA     A       ;IL PULSANTE È CHIUSO (0)?
JP      DONE    ;SÌ, SALTA A DONE
```

L'istruzione IN non influenza i flag; quindi occorre l'istruzione ORA per porre ad 1 i flag.

```
Bit 6
IN      PORT    ;LEGGI POSIZIONE PULSANTE
ADD     A       ;IL PULSANTE È CHIUSO (0)?
JP      DONE    ;SÌ, SALTA A DONE
```

L'istruzione RAL non può essere impiegata perchè non influenza il bit Segno.

Compito 2: Conta il numero di chiusure del pulsante incrementando la locazione di memoria 40.

RIMBALZO DELL'INTERRUTTORE

Per contare il numero di volte che il pulsante è stato premuto, occorre essere sicuri che ogni chiusura origini una singola transizione. Comunque un pulsante meccanico non produce una singola transizione per ogni chiusura poichè i contatti meccanici rimbalzano avanti ed indietro prima di assumere le loro posizioni finali. È possibile l'impiego dell'hardware per eliminare il rimbalzo oppure si può controllare con il software.

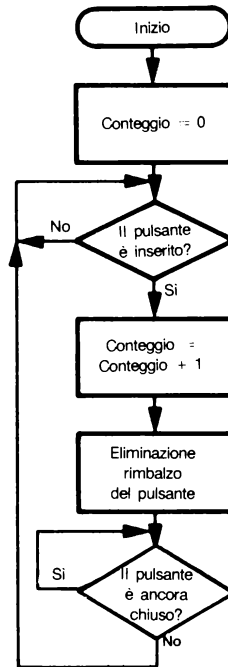
ELIMINAZIONE RIMBALZO IN SOFTWARE

Il programma può eliminare il rimbalzo del pulsante mediante l'attesa che esso assuma una chiusura. Il tempo richiesto è chiamato tempo di eliminazione rimbalzo ed è una caratteristica del pulsante. Esso tipicamente è di pochi millisecondi. Il programma non dovrebbe esaminare il pulsante durante questo periodo perché potrebbe considerare erroneamente il rimbalzo come una nuova chiusura. Il programma può entrare in una routine di ritardo come quelle precedentemente descritte oppure può semplicemente eseguire altri compiti per un tempo ben specificato.

Anche dopo l'eliminazione del rimbalzo il programma deve ancora aspettare per la chiusura attuale per terminare prima guardando per una nuova chiusura. Questa procedura evita che la stessa chiusura venga conteggiata più di una volta. Il seguente programma impiega un ritardo di software di 1 ms per eliminare il rimbalzo del pulsante. Si può provare a variare il ritardo oppure ad eliminarlo interamente per vedere cosa accade. Per eseguire questo programma è necessario fare entrare la subroutine di ritardo nella memoria iniziando dalla locazione 30.

Problema Campione:
Premendo il pulsante dieci volte dopo l'inizio del programma si dovrebbe avere

(40) = 0A

Diagramma di Flusso:

Programma Sorgente:

```
        LXI    H,40H    ;CONTEGGIO = 0
        MVI    M,0
CHKCL:  IN     PORT     ;IL PULSANTE È CHIUSO (0)?
        ANI    MASK
        JNZ    CHKCL    ;NO, ATTENDI
        INR    M        ;SÌ, CONTEGGIO = CONTEGGIO + 1
        MVI    A,1
        CALL   DELAY    ;ELIMINA RIMBALZO PULSANTE ASPETTANDO PER 1 MS
CHKOP:  IN     PORT     ;IL PULSANTE È ANCORA CHIUSO (0)?
        ANI    MASK
        JZ     CHKOP    ;SÌ, ATTENDI
        JMP    CHKCL    ;NO, GUARDA PER LA SUCCESSIVA CHIUSURA
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		LXI	H,40H	21
01				40
02				00
03		MVI	M,0	36
04				00
05	CHKCL:	IN	PORT	DB
06				PORT
07		ANI	MASK	E6
08				MASK
09		JNZ	CHKCL	C2
0A				05
0B				00
0C		INR	M	34
0D		MVI	A,1	3E
0E				01
0F		CALL	DELAY	CD
10				30
11				00
12	CHKOP:	IN	PORT	DB
13				PORT
14		ANI	MASK	E6
15				MASK
16		JZ	CHKOP	CA
17				12
18				00
19		JMP	CHKCL	C3
1A				05
1B				00

Si noti che le tre istruzioni inizianti con la label CHKOP sono impiegate per determinare quando l'interruttore riapre.

Uno switch a ginocchiera (SPDT)

Scopo: Serve per realizzare l'interfaccia di uno switch unipolare a due vie (SPDT) con un microprocessore 8080. La switch a ginocchiera è un dispositivo meccanico che può essere nella posizione normalmente chiusa (NC) oppure in quella normalmente aperta (NO).

ELIMINAZIONE RIMBALZO CON GATE NAND AD ACCOPPIAMENTO INCROCIATO

La Figura 11-8 mostra la circuiteria per realizzare l'interfaccia dello switch. Come per il pulsante, lo switch impiega un bit di una porta d'ingresso 8212 non latch che serve come buffer indirizzabile. Diversamente dal pulsante, lo switch può permanere in entrambe le posizioni; tipici esercizi di programma sono la determinazione della posizione dell'interruttore e la verifica se la posizione è variata. Una coppia di gate NAND ad accoppiamento incrociato (Vedere Figura 11-9) può eliminare l'elasticità di uno switch meccanico.

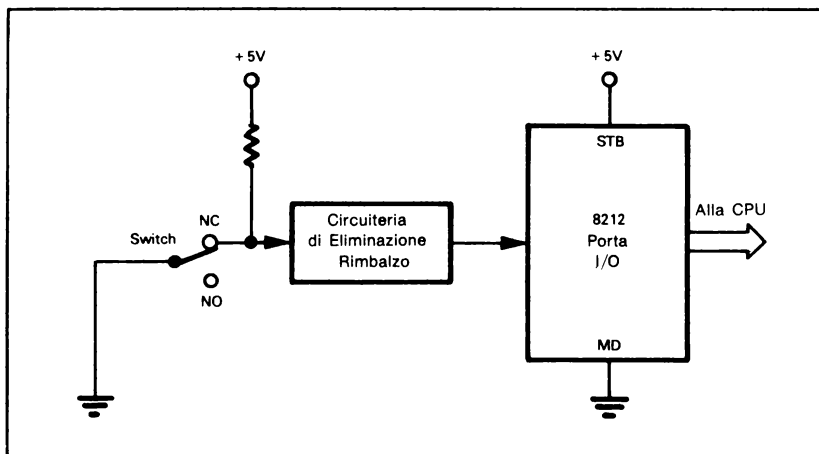



Figura 11-8
Un Circuito Switch

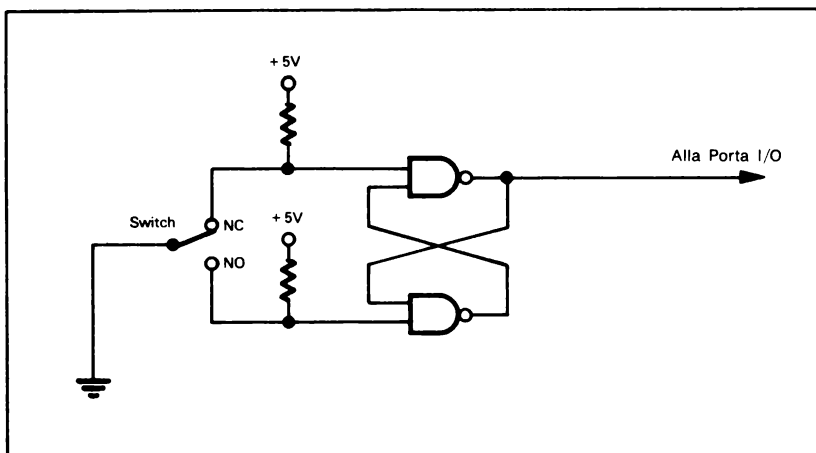



Figura 11-9
Un Circuito Di Eliminazione Rimbalzo Basato Su Porte NAND Ad Accoppiamento-Incrociato

Detto circuito produce una fase singola in risposta alla variazione della posizione dello switch anche se lo switch oscilla prima di stabilirsi in una nuova posizione.

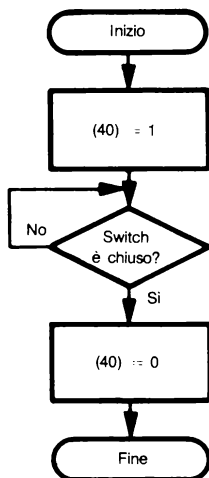
Esempi di Programmazione:

Verranno eseguiti due compiti che coinvolgono questo circuito. Essi sono:

1) Osservazione dell'uscita dello switch ed azzeramento di una locazione di memoria quando i contatti normalmente aperti dello switch sono chiusi.
2) Osservazione dell'uscita dello switch ed azzeramento di una locazione di memoria quando l'uscita dello switch cambia.

Compito 1: Attesa per la chiusura dello switch.

La locazione di memoria 40 permane ad 1 fino a che lo switch è chiuso e poi viene azzerata, cioè il processore pone la locazione 40 ad 1, attende che lo switch venga chiuso e poi azzera la stessa locazione. Lo switch funziona come un comando MARCIA/ARRESTO poichè il processore non procederà fino a che lo switch è chiuso.

Diagramma di Flusso:

Programma Sorgente:

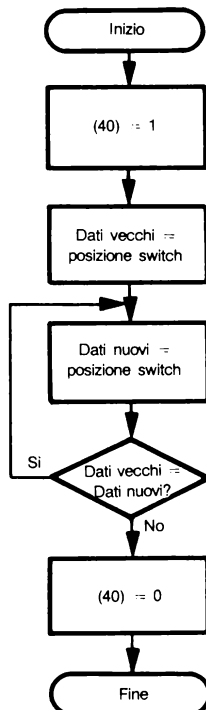
```
        LXI    H,40H    ;ASSEGNAZIONE = 1
        MVI    M,1
WAITC:  IN     PORT     ;LEGGE POSIZIONE SWITCH
        ANI    MASK     ;LO SWITCH È CHIUSO (0)?
        JNZ    WAITC    ;NO, ATTENDI
        DCR    M        ;SÌ, ASSEGNAZIONE = 0
DONE:   JMP    DONE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		LXI	H,40H	21
01				40
02				00
03		MVI	M,1	36
04				01
05	WAITC:	IN	PORT	DB
06				PORT
07		ANI	MASK	E6
08				MASK
09		JNZ	WAITC	C2
0A				05
0B				00
0C		DCR	M	35
0D	DONE:	JMP	DONE	C3
0E				0D
0F				00

Compito 2: Attesa per il cambiamento di posizione dello switch.

La locazione di memoria 40 rimane ad 1 fino a che la posizione dello switch cambia, cioè il processore attende fino a che lo switch cambia posizione, poi azzera la locazione di memoria 40.

Diagramma di Flusso:

Programma Sorgente:

```
        LXI     H,40H   ;ASSEGNAZIONE = 1
        MVI     M,1     ;ACCETTA LA VECCHIA POSIZIONE DELLO SWITCH
        IN      PORT
        ANI     MASK
        MOV     B,A
SRCH:   IN      PORT    ;ACCETTA LA NUOVA POSIZIONE DELLO SWITCH
        ANI     MASK
        CMP     B       ;LA VECCHIA E LA NUOVA POSIZIONE SONO UGUALI?
        JZ      SRCH    ;SÌ, ATTENDI
        DCR     M       ;NO, ASSEGNAZIONE = 0
DONE:   JMP     DONE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)	Contenuti di Mem. (Esadec.)
00	LXI H,40H	21
01		40
02		00
03	MVI M,1	36
04		01
05	IN PORT	DB
06		PORT
07	ANI MASK	E6
08		MASK
09	MOV B,A	47
0A	SRCH: IN PORT	DB
0B		PORT
0C	ANI MASK	E6
0D		MASK
0E	CMP B	B8
0F	JZ SRCH	CA
10		0A
11		00
12	DCR M	35
13	DONE: JMP DONE	C3
14		13
15		00

Le istruzioni SUB A oppure XRA B potrebbero sostituire CMP B in questo programma. Entrambe queste istruzioni potrebbero comunque cambiare i contenuti dell'Accumulatore. Se diversi switch sono commessi alla PORT allora XRA B può essere più conveniente di CMP B, poichè XRA B produrrebbe un bit 1 per ogni switch che cambia stato. Come si potrebbe riscrivere questo programma in modo da eliminare il rimbalzo via software?

Uno Switch a Posizione - Multipla (Rotante, Selettore oppure Thumbwheel)

Scopo: Realizzare l'interfaccia di uno switch a posizione multipla con un microprocessore 8080. Lo switch fornisce una connessione a terra nel terminale corrispondente alla posizione dello switch.

La Figura 11-10 mostra la circuiteria richiesta per realizzare l'interfaccia di uno switch ad 8-posizioni. Lo switch impiega tutti gli 8 bit di una porta d'ingresso 8212. Compiti tipici sono la determinazione della posizione dello switch ed il controllo se la posizione è cambiata. Occorre considerare i seguenti casi particolari:

1) Lo switch è temporaneamente tra due posizioni cosicchè nessun terminale è messo a terra.
2) Lo switch non ha ancora raggiunto la sua posizione finale.

La prima di queste situazioni può essere trattata attendendo fino a che l'ingresso è diverso da tutti uni, cioè fino a che una posizione dello switch è messa a terra. La seconda situazione può essere trattata esaminando ancora lo switch dopo un ritardo (per esempio 1 o 2 secondi) ed accettando l'ingresso solo se è ripetuto. Questo ritardo normalmente non influenzerà la risposta del sistema allo switch. È possibile anche usare un altro switch (cioè uno switch di carico LOAD) per dire al processore che lo switch selettore può essere letto.

Esempi di Programmazione:
Si eseguiranno due compiti che coinvolgono il circuito di Figura 11-10. Questi sono:

1) Ispezionare l'uscita dello switch fino ad una condizione stabile. Rivelata questa viene esaminata la posizione dello switch ed immagazzinato il numero binario equivalente in una ben definita locazione di memoria.
2) Attesa di un cambiamento della posizione dello switch quindi immagazzinamento della nuova posizione in una locazione di memoria.

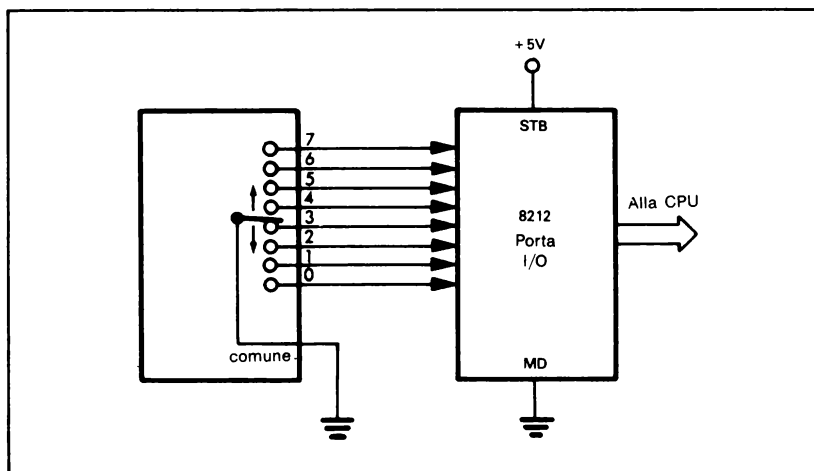



Figura 11-10
Uno Switch A Posizioni Multiple

Se switch si trova in una particolare posizione, il terminale corrispondente è collegato a terra tramite la linea comune. I resistori di pullup sono normalmente necessari in tutte le linee di ingresso per evitare problemi causati dal rumore.

Compito 1: Determina la posizione dello switch.

Il programma attende che lo switch si trovi in una specifica posizione e pone il numero corrispondente a quella posizione nella locazione di memoria 40.
La seguente tabella pone in relazione i dati di ingresso con la posizione dello switch per la Figura 11-10.

Posizione Switch	Dati d'Ingresso	
	Binario	Esadecimale
0	11111110	FE
1	11111101	FD
2	11111011	FB
3	11110111	F7
4	11101111	EF
5	11011111	DF
6	10111111	BF
7	01111111	7F

Tabella 11-1. Dati Di Ingresso, Rispetto Alla Posizione Switch.

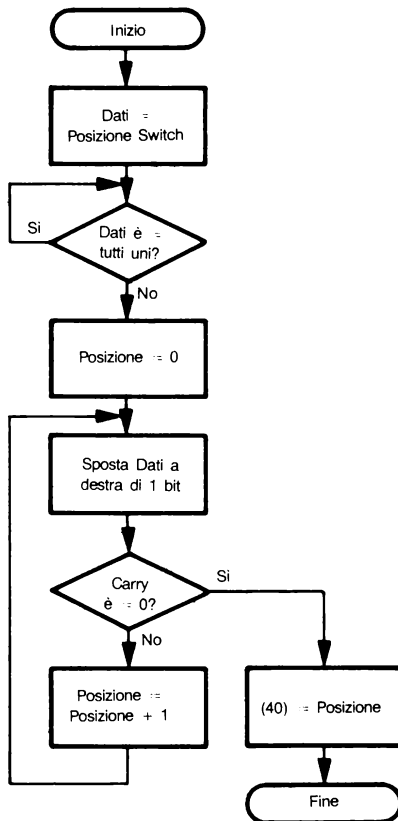
Si noti la inefficienza di questo schema che richiede 8 bit per distinguere 8 posizioni diverse.

IMPIEGO DI UN CODIFICATORE DIGITALE

Un codificatore digitale potrebbe ridurre il numero di bit richiesti. La Figura 11-11 mostra un circuito impiegante il codificatore 74LS148 TTL da 8 a 3. Si collegano le uscite dello switch in ordine inverso poichè il dispositivo 74LS148 ha uscite attive basse. Il risultato dell'impiego del circuito codificatore è la rappresentazione a 3 bit della posizione dello switch. Molti switch hanno incorporato codificatori o matrici di diodi in modo che essi producono automaticamente un'uscita codificata, normalmente BCD.

Figura 11-11. Uno Switch A Posizioni Multiple Con Codificatore

Il codificatore fornisce uscite che sono attive basse. Si noti, per esempio, che l'uscita dalla posizione 5 dello switch a posizioni multiple è connesso ad T_2 del codificatore. Il codificatore accetta la rappresentazione a tre bit binari (000) di 2 e produce 101, che è la rappresentazione di 2 attiva bassa.

Diagramma di Flusso:

Programma Sorgente:

```
CHKSW:  IN     PORT       ;ACCETTA DATI SWITCH
        CPI    11111111B  ;LO SWITCH È IN UNA POSIZIONE?
        JZ     CHKSW      ;NO, ATTENDI PER POSIZIONE
        MVI    B,0        ;POSIZIONE = 0
CHPOS:  RAR               ;IL BIT SUCCESSIVO CORRISPONDE
                          ;POSIZIONE A TERRA?
        JNC    DONE       ;SÌ, TROVATA POSIZIONE SWITCH
        INR    B          ;NO, POSIZIONE = POSIZIONE + 1
        JMP    CHPOS
DONE:   LXI    H,40H      ;IMMAGAZZINA POSIZIONE SWITCH
        MOV    M,B
HERE:   JMP    HERE
```

Programma Oggetto:

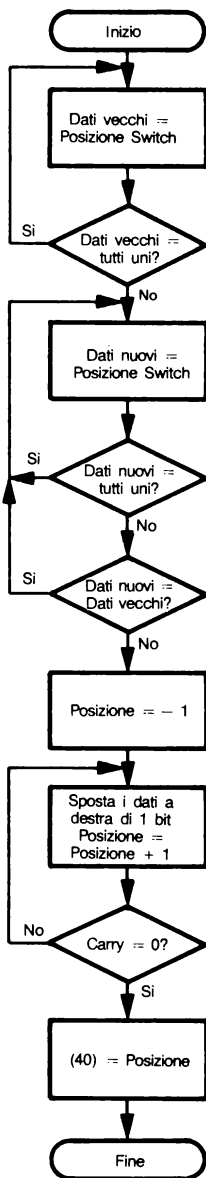
Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00	CHKSW:	IN	PORT	DB
01				PORT
02		CPI	11111111B	FE
03				FF
04		JZ	CHKSW	CA
05				00
06				00
07		MVI	B,0	06
08				00
09	CHPOS:	RAR		1F
0A		JNC	DONE	D2
0B				11
0C				00
0D		INR	B	04
0E		JMP	CHPOS	C3
0F				09
10				00
11	DONE:	LXI	H,40H	21
12				40
13				00
14		MOV	M,B	70
15	HERE:	JMP	HERE	C3
16				15
17				00

Si supponga che uno switch imperfetto oppure una porta I/O difettosa faccia in modo che l'ingresso sia sempre OFF_{16}. Come si potrebbe cambiare il programma in modo che riveli questo errore?

Nel programma sorgente è presente il salto incondizionato JMP CHPOS. Si saprebbero ristrutturare le condizioni iniziali in modo che questo salto non necessario venga eliminato?

Compito 2: Attesa per variazione della posizione dello switch.

Il programma attende che la posizione dello switch vari e carica la nuova posizione (decodificata) nella locazione di memoria 40. Il programma ignora stati intermedi in cui lo switch non occupa alcuna posizione.

Diagramma di Flusso:

Programma Sorgente:

```
CHFST:  IN    PORT       ;ACCETTA DATI SWITCH
        CPI   11111111B  ;LO SWITCH È IN UNA POSIZIONE?
        JZ    CHFST      ;NO, ATTENDI PER POSIZIONE
        MOV   B,A
CHSEC:  IN    PORT       ;ACCETTA NUOVI DATI SWITCH
        CPI   11111111B  ;LO SWITCH È IN UNA POSIZIONE?
        JZ    CHSEC      ;NO, ATTENDI PER POSIZIONE
        CMP   B          ;LA POSIZIONE È LA STESSA DI PRIMA?
        JZ    CHSEC      ;SÌ, ATTENDI
        MVI   B,0FFH     ;POSIZIONE = -1
CHPOS:  RAR              ;IL BIT SUCCESSIVO COINCIDE POSIZIONE A TERRA?
        INR   B          ;POSIZIONE = POSIZIONE + 1
        JC    CHPOS      ;NO, SEGUITA RICERCA POSIZIONE A TERRA
DONE:   LXI   H,40H      ;IMMAGAZZINA POSIZIONE SWITCH
        MOV   M,B
HERE:   JMP   HERE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00	CHFST:	IN	PORT	DB
01				PORT
02		CPI	11111111B	FE
03				FF
04		JZ	CHFST	CA
05				00
06				00
07		MOV	B,A	47
08	CHSEC:	IN	PORT	DB
09				PORT
0A		CPI	11111111B	FE
0B				FF
0C		JZ	CHSEC	CA
0D				08
0E				00
0F		MVI	B,0FFH	06
10				FF
11	CHPOS:	RAR		1F
12		INR	B	04
13		JC	CHPOS	DA
14				11
15				00
16	DONE:	LXI	H,40H	21
17				40
18				00
19		MOV	M,B	70
1A	HERE:	JMP	HERE	C3
1B				1A
1C				00

Un LED Singolo

Scopo: Realizzare l'interfaccia di un singolo diodo emettitore di luce (LED) con un microprocessore 8080. Il LED può essere connesso in modo tale che venga commutato ON da uno '0' ovvero da un '1'.

La Figura 11-12 mostra la circuiteria per illuminare un LED. Il LED emette luce quando il suo anodo è positivo rispetto al catodo (Figura 11-12 a). Così la circuiteria può fare illuminare il LED collegando a terra il catodo avendo l'alimentazione del calcolatore ad '1' per l'anodo (Figura 11-12 b) oppure mantenendo l'anodo a + 5 Volt avendo l'alimentazione del calcolatore a '0' per il catodo (Figura 11-12 c). Impiegando il catodo è fatto funzionare da corrente pulsante da 10 a 50 m A circa applicata qualche centinaio di volte al secondo. Il LED ha un tempo di accensione molto breve (dell'ordine del microsecondo) e cosi è molto adatto al multiplexing cioè l'impiego di diversi dispositivi comandati da una sola porta. I circuiti LED naturalmente richiedono periferiche o driver a transistor e resistori di limitazione della corrente.

a) Circuiteria base per LED. Il resistore R dovrebbe limitare la massima corrente a 50 mA e la corrente media a 10 mA.

b) Realizzazione dell'interfaccia di un LED con logica positiva. Un '1' logico dalla CPU accende il LED.

c) Realizzazione dell'interfaccia di un LED con logica negativa. Uno '0' logico dalla CPU accende il LED. Il driver oppure la CPU possono invertire i livelli logici. Si noti che la porta d'uscita 8212 funziona da latch quando MD è connesso a + 5V.

Figura 11-12
Realizzazione Dell'Interfaccia Di Un LED

Esempi di Programmazione:

Il programma posiziona acceso o spento un singolo LED.

Compito 1: Invia un '1' logico al LED (cioè accende un display positivo oppure spegne un display negativo).

Programma Sorgente:

```
       MVI   A,MASKP
       OUT   PORT     ;DATI AI LED
       STA   40H      ;CONSERVA COPIA DATI DI USCITA
HERE:  JMP   HERE
```

(dati aggiornati nella locazione 40)

```
       LXI   H,40H    ;CARICA PUNTATORE A DATI
       MOV   A,M      ;ACCETTA COPIA DATI DI USCITA
       ORI   MASKP    ;BIT LED = 1
       OUT   PORT     ;DATI AL LED
       MOV   M,A      ;CONSERVA COPIA DATI DI USCITA
HERE:  JMP   HERE
```

MASKP ha un '1' nella posizione del bit assegnato al LED che si vuole illuminare. Ci sono zeri in tutte le altre posizioni di bit. Facendo l'OR con MASKP non vengono influenzate le altre posizioni di bit. Si noti che la CPU non può leggere direttamente i dati dalla porta di uscita cosicchè è necessaria una copia.

Programma Oggetto: (forma i dati inizialmente)

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)	Contenuti di Mem. (Esadec.)
00	MVI A,MASKP	3E
01		MASKP
02	OUT PORT	D3
03		PORT
04	STA 40H	32
05		40
06		00
07	HERE: JMP HERE	C3
08		07
09		00

Programma Oggetto: (aggiorna i dati nella locazione 40H)

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)	Contenuti di Mem. (Esadec.)
00	LXI H,40H	21
01		40
02		00
03	MOV A,M	7E
04	ORI MASKP	F6
05		MASKP
06	OUT PORT	D3
07		PORT
08	MOV M,A	77
09	HERE: JMP HERE	C3
0A		09
0B		00

Compito 2: Invia uno 0' logico al LED (così spegne un display positivo e ne accende uno negativo).

Programma Sorgente: (forma i dati inizialmente)

```
        MVI     A,MASKN
        OUT     PORT     ;DATI AI LED
        STA     40H      ;CONSERVA COPIA DEI DATI DI USCITA
HERE:   JMP     HERE
```

(aggiorna dati nella locazione 40H)

```
        LXI     H,40H    ;CARICA PUNTATORE A DATI
        MOV     A,M      ;ACCETTA COPIA DATI DI USCITA
        ANI     MASKN    ;BIT LED = 0
        OUT     PORT     ;DATI AI LED
        MOV     M,A      ;CONSERVA COPIA DEI DATI DI USCITA
HERE:   JMP     HERE
```

MASKN ha un bit '0' nella posizione del LED e tutti uni nelle altre posizioni. Eseguendo l'AND logico con MASKN non vengono influenzate le altre posizioni di bit.

Programma Oggetto: (forma i dati inizialmente)

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)	Contenuti di Mem. (Esadec.)
00	MVI A,MASKN	3E
01		MASKN
02	OUT PORT	D3
03		PORT
04	STA 40H	32
05		40
06		00
07	HERE: JMP HERE	C3
08		07
09		00

Programma Oggetto: (aggiorna i dati nella locazione 40H)

00	LXI H,40H	3A
01		40
02		00
03	MOV A,M	7E
04	ANI MASKN	E6
05		MASKN
06	OUT PORT	D3
07		PORT
08	MOV M,A	77
09	HERE: JMP HERE	C3
0A		09
0B		00

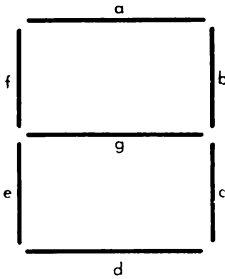
Display LED a 7-Segmenti

Scopo: Realizzare l'interfaccia di un display LED a 7 segmenti con un microprocessore 8080. Il display può essere ad anodo comune (logica negativa) oppure a catodo comune (logica positiva).

DISPLAY AD ANODO COMUNE ED A CATODO COMUNE

La Figura 11-13 mostra la circuiteria richiesta per realizzare l'interfaccia di un display a 7 segmenti. Ogni elemento è un singolo LED. Ci sono due metodi per connettere i LED. Uno è quello di congiungere tutti i catodi a massa (Vedere Figura 11-14 a), questo è un «display a catodo comune» ed un '1' logico illumina un segmento. L'altro modo è quello di congiungere assieme tutti gli anodi ad un potenziale positivo di alimentazione: questo è un «display ad anodo comune» ed uno '0' logico applicato al catodo illumina un segmento. Così il display a catodo comune impiega la logica negativa. Entrambi i tipi di display richiedono opportuni driver e resistori.

Figura 11-13
Realizzazione Dell'Interfaccia Di Un Display A 7 Segmenti

La linea COMUNE del display è collegata a massa in un caso ed a + 5 Volt nell'altro. Il display è disposto in questo modo:

L'ottavo bit può controllare un LED rappresentante il punto decimale.

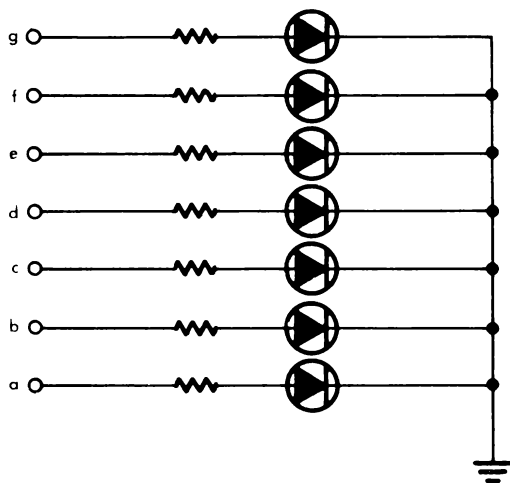
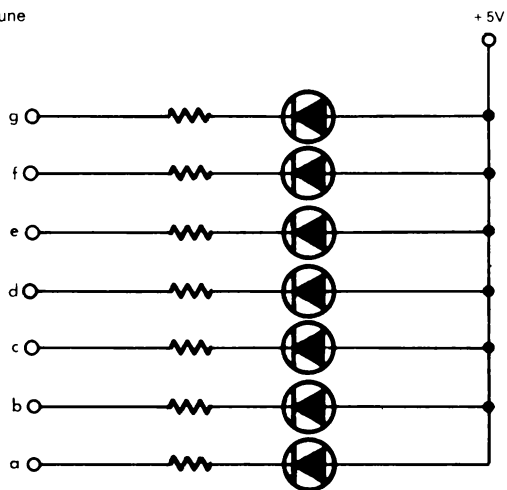



Figura 11-14
Organizzazione Di Un Display

RAPPRESENTAZIONI A 7 SEGMENTI

Si noti che il display a 7 segmenti è largamente impiegato perchè contiene il numero più piccolo di parti luminose controllate separatamente in grado di fornire la rappresentazione facilmente riconoscibile di tutte le cifre decimali (vedere la Figura 11-15 e la Tabella 11-2). I display a 7 segmenti possono anche fornire alcune lettere e segni (Vedere Tabella 11-3). Rappresentazioni migliori richiedono un numero sostanzialmente maggiore di elementi del display e più circuiteria. La popolarità dei display a 7-segmenti ha portato ad una larga disponibilità di decodificatori/driver a 7 segmentij (per esempio 74LS47, 74LS48, 74LS49 e 4511). Alcuni dispositivi sono anche dotati dell'ingresso LAMP TEST (che accende tutti gli elementi del display per scopi di controllo) ed azzeramento degli ingressi e delle uscite (per eliminare gli zeri non significativi).

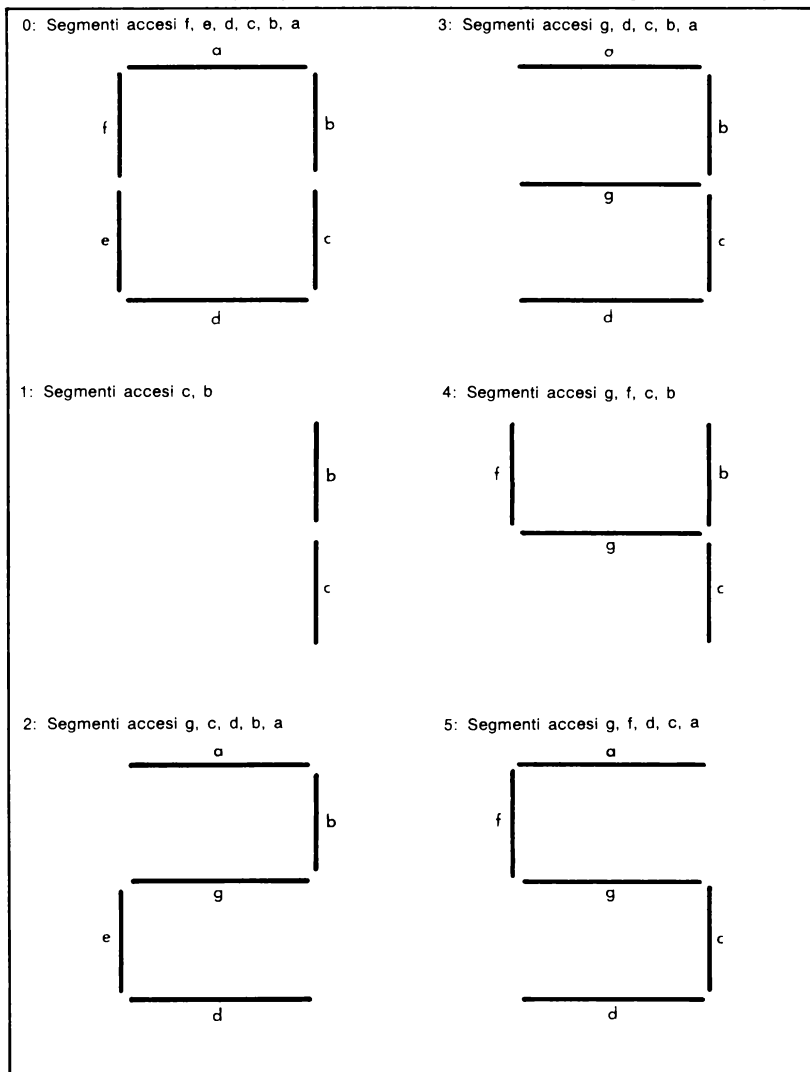



Figura 11-15
Rappresentazione A 7 Segmenti Delle Cifre Decimali

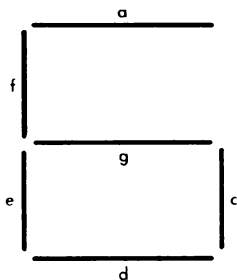
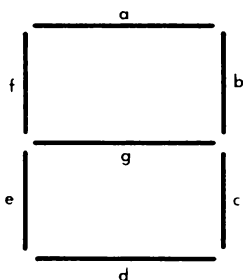
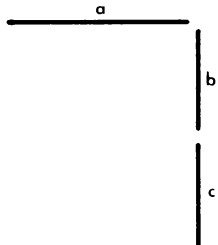
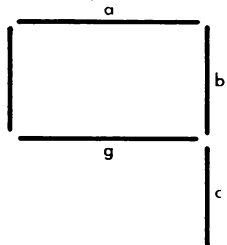



Figura 11-15
Rappresentazione A 7 Segmenti Delle Cifre Decimali (seguito)

Tabella 11-2
Rappresentazione A 7 Segmenti Di Numeri Decimali

Numero	Rappresentazione Esadecimale	
	Catodo Comune	Anodo Comune
0	3F	40
1	06	79
2	5B	24
3	4F	30
4	66	19
5	6D	12
6	7D	02
7	07	78
8	7F	00
9	67	18

Il bit 7 è sempre 0 e gli altri sono g, f, e, d, c, b, a in ordine decrescente di significato.

Tabella 11-3
Rappresentazione A 7 Segmenti di Lettere e Simboli

Lettere Maiuscole

Lettera	Rappresentazione Esadecimale	
	Catodo Comune	Anodo Comune
A	77	08
C	39	46
E	79	06
F	71	0E
H	76	09
I	06	79
J	1E	61
L	38	47
O	3F	40
P	73	0C
U	3E	41
Y	66	19

Lettere Minuscole e Caratteri Speciali

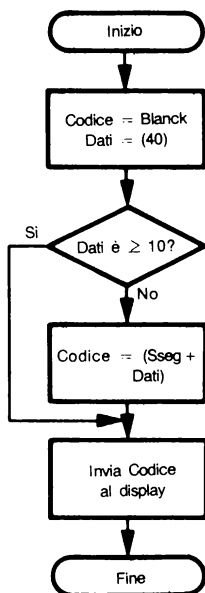
Lettera	Rappresentazione Esadecimale	
	Catodo Comune	Anodo Comune
b	7C	03
c	58	27
d	5E	21
h	74	0B
n	54	2B
o	5C	23
r	50	2F
u	1C	63
-	40	3F
?	53	2C

Esempi di Programmazione:

Mostra i contenuti della locazione di Memoria 40 su un display a 7 segmenti se 40 contiene una cifra decimale. Diversamente spegne il display.

Problemi Campione:

a. (40) = 05
Risultato = 5 sul display
b. (40) = 66
Risultato = Carattere blank sul display

Diagramma di Flusso:

Programma Sorgente:

```
        MVI    B,BLANK ;ACCETTA IL CODICE DI BLANK
        LDA    40H     ;ACCETTA DATI
        CPI    10      ;DATI È >10?
        JNC    DSPLY   ;SÌ, MOSTRA CARATTERE BLANK
        LXI    D,SSEG  ;BASE DELLA TABELLA A 7 SEGMENTI
        MVI    H,0
        MOV    L,A     ;TRASFORMA DATI IN INDICE A 16 BIT
        DAD    D       ;INDICE TABELLA
        MOV    B,M     ;ACCETTA IL CODICE A 7 SEGMENTI
DSPLY:  MOV    A,B
        OUT    PORT    ;CODICE AL DISPLAY
HERE:   JIMP   HERE
```

BLANK corrisponde a 00 per un display a catodo comune, FF per un display ad anodo comune. Una procedura alternativa potrebbe essere quella di mettere il codice blank alla fine della tabella e sostituire tutti i valori di dati errati con 10, cioè:

```
        LDA    40H     ;ACCETTA DATI
        CPI    10      ;DATI È >10?
        JC     CONVRT  ;NO, VAI ALLA CONVERSIONE A 7 SEGMENTI
        MVI    A,10    ;SÌ, ACCETTA INDICE PER CODICE BLANK
CNVRT:  LXI    D,SSEG  ;BASE DELLA TABELLA A 7 SEGMENTI
```

La Tabella SSEG è la rappresentazione sia per catodo comune che per anodo comune dalla Tabella 11-2.

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		MVI	B,BLANK	06
01				BLANK
02		LDA	40H	3A
03				40
04				00
05		CPI	10	FE
06				0A
07		JNC	DSPLY	D2
08				12
09				00
0A		LXI	D,SSEG	11
0B				18
0C				00
0D		MVI	H,0	26
0E				00
0F		MOV	L,A	6F
10		DAD	D	19
11		MOV	B,M	46
12		MOV	A,B	78
13	DSPLY:	OUT	PORT	D3
14				PORT
15	HERE:	JMP	HERE	C3
16				15
17				00
18-21		SSEG		

PROBLEMI

1) Un Pulsante di Accensione - Spegnimento (ON-OFF)

Scopo: Ad ogni chiusura di un pulsante complementa (inverte) tutti i bit della locazione di memoria 40. La locazione inizialmente contiene tutti zeri. Il programma potrebbe esaminare continuamente il pulsante e complementare la locazione 40 ad ogni chiusura.

Caso Campione:

La locazione 40 inizia con zero.

La prima chiusura del pulsante cambia la locazione 40 ad FF (esadec.), la seconda fa ritornare a zero, la terza ad FF (esadec.), ecc.. Si assuma che l'eliminazione del rimbalzo del pulsante sia realizzata via hardware. Come si potrebbe comprendere l'eliminazione del rimbalzo nel programma che si vuole realizzare?

2) Eliminazione del Rimbalzo di uno Switch via Software

Scopo: Elimina il rimbalzo di uno switch meccanico aspettando fino a che due letture, intervallate da un tempo di eliminazione rimbalzo danno lo stesso risultato. Si assuma che il tempo di eliminazione rimbalzo (in ms) sia contenuto nella locazione di memoria 40 e la posizione dello switch sia contenuta nella locazione 41.

Problema Campione:

(40) = 03 cioè il programma attende 3ms tra le due letture.

3) Controllo di uno Switch Rotante

Scopo: Un altro switch serve come switch di carico per uno switch rotante a quattro posizioni non codificate. La CPU attende che lo switch di carico sia chiuso e poi legge la posizione dello switch rotante. Questa procedura permette all'operatore di selezionare una posizione finale per lo switch rotante prima che la CPU sia indotta alla lettura. Il programma potrebbe localizzare la posizione dello switch rotante nella locazione di memoria 40. Si realizza l'eliminazione rimbalzo dello switch di CARICO via software.

Problema Campione:

Si ponga lo switch rotante nella posizione 2. Si chiuda lo switch di CARICO.

Risultato = (40) = 02

4) Registrazione Luminosa delle Posizioni di uno Switch

Scopo: Un set di otto switch potrebbero avere la loro rappresentazione in altrettanti LED, cioè se lo switch è chiuso ('0') il LED corrispondente è spento; altrimenti il LED è acceso. Si assuma che la parte di uscita della CPU sia connessa ai catodi dei LED.

Problema Campione:

SWITCH 0 CHIUSO
SWITCH 1 APERTO
SWITCH 2 CHIUSO
SWITCH 3 APERTO
SWITCH 4 APERTO
SWITCH 5 CHIUSO
SWITCH 6 CHIUSO
SWITCH 7 APERTO

Risultato:

LED 0 SPENTO
LED 1 ACCESO
LED 2 SPENTO
LED 3 ACCESO
LED 4 ACCESO
LED 5 SPENTO
LED 6 SPENTO
LED 7 ACCESO

Come si potrebbe modificare il programma in modo che lo switch connesso al bit 7 della Porta 2 determini se il display è attivo oppure no (cioè se lo switch di controllo è chiuso il display riflette gli altri switch; se lo switch di controllo è aperto, il display è spento)? Uno switch di controllo è comodo quando i display devono essere indipendenti dall'operatore, come in un aereoplano.

Come si potrebbe cambiare il programma se lo switch di controllo fosse un pulsante di accensione-spegnimento, cioè tale che ad ogni chiusura inverte lo stato precedente del display? Si assuma che il display inizi da uno stato attivo e che il programma esamini e realizzi l'eliminazione rimbalzo dello switch prima dell'attivazione del display.

5) Conteggio su un display a 7 Segmenti

Scopo: Il programma potrebbe contare da 0 a 9 continuamente su un display a 7 segmenti, partendo da zero.

Si provino diverse lunghezze di timing e si osservi cosa succede. Quando il conteggio diventa visibile? Cosa succede se il display è blank per una parte del tempo?

DISPOSITIVI DI I/O PIÙ COMPLESSI

I dispositivi di I/O più complessi si differenziano dalle semplici tastiere, switch e display perchè:

1) Essi operano con velocità dati più elevate.
2) Possono avere propri clock interni e timing.
3) Producono informazioni di stato e richiedono informazioni di controllo come pure il trasferimento dei dati.

Velocità dati più elevate significa che si può maneggiare questi dispositivi in modo random. Se il processore non fornisce il servizio appropriato, il dispositivo di I/O può omettere dati d'ingresso oppure produrre dati di uscita errati. Quindi è necessario lavorare sotto limiti più stringenti rispetto al caso di dispositivi lenti. Gli interrupt sono un metodo conveniente per maneggiare dispositivi di I/O complessi come si vedrà al capitolo 12.

SINCRONIZZAZIONE CON DISPOSITIVI DI I/O

Le periferiche come tastiere, telescriventi, cassette e floppy disk normalmente sono dotate di un timing interno. Questi dispositivi generalmente forniscono stringhe o blocchi di dati separati da specifici intervalli di timing. Il computer deve sincronizzare le operazioni iniziali di ingresso ed uscita con il clock periferico e quindi fornire un intervallo opportuno tra operazioni successive. Un semplice ciclo di ritardo come quello mostrato precedentemente produrrà l'intervallo di timing. La sincronizzazione può richiedere una o più procedure seguenti:

1) Ricerca di una transizione su un clock od una linea strobe forniti da una periferica per scopi di timing. Un metodo semplice è quello di congiungere la linea strobe al bit più significativo di una porta di ingresso ed osservare un cambiamento del flag segno.
2) Ricerca del centro dell'intervallo di tempo durante il quale i dati sono validi. Sarebbe preferibile determinare il valore dei dati al centro dell'impulso piuttosto che al bordo dove i dati possono essere variabili. La ricerca del centro richiede un ritardo di mezzo intervallo di trasmissione (tempo di bit) dopo il gradino. Considerando i dati significa anche piccoli errori di timing hanno poca influenza sull'accuratezza della ricezione.
3) Riconoscimento di un codice di partenza speciale. Questo è facile se il codice è un singolo bit oppure se si hanno alcune informazioni di timing. La procedura è molto più complessa se il codice è lungo e potrebbe iniziare ad un istante generico. Scorrimenti saranno necessari per determinare dove il trasmettitore ha iniziato i suoi bit, caratteri o messaggi (questo viene spesso chiamato ricerca della «framing» corretta).
4) Campionamento dei dati. Prendendo diversi campioni dei dati si riduce la probabilità di ricezione non corretta a causa della rumorosità delle linee. La logica maggioritaria può essere usata per decidere sul valore effettivo dei dati.

Naturalmente la ricezione è molto più difficoltosa della trasmissione poichè la periferica controlla la ricezione ed il computer deve interpretare le informazioni di timing generate dalla periferica. In trasmissione il computer fornisce il timing ed il formato opportuno per una particolare periferica.

INFORMAZIONI DI CONTROLLO E DI STATO

Le periferiche possono richiedere o fornire una grande quantità di informazioni oltre ai dati ed al timing. L'altra informazione trasmessa dal computer è considerata «informazione di controllo»; è possibile selezionare il modo di operare, l'inizio o l'arresto di processi, registrare clock, abilitare buffer, scegliere formati o protocolli, fornire display all'operatore, eseguire operazioni di conteggio oppure identificare il tipo o la priorità dell'operazione. L'altra informazione trasmessa dalla periferica è considerata «informazione di stato»; essa può indicare il modo di funzionamento, la mancanza di ready (pronto) dei dispositivi, la presenza di condizioni di errore, il formato o protocollo da usare ed altre condizioni o stati.

Il computer tratta le informazioni di controllo e di stato esattamente come dati. Questa informazione spesso agisce come dati da una periferica lenta poichè essa raramente cambia anche se i dati effettivi possono essere trasferiti ad alta velocità. L'informazione può essere in bit singoli, digit, parole oppure parole multiple. I Bit singoli o campi corti vengono spesso combinati e trattati da una singola porta d'ingresso o d'uscita.

Combinando l'informazione di stato e di controllo in byte si riduce il numero totale di indirizzi della porta I/O richiesti dalle periferiche. Comunque la combinazione significa che i bit di ingresso di stato individuali devono essere interpretati separatamente ed i bit di controllo di uscita devono essere determinati separatamente. La procedura per l'isolamento dei bit di stato è la seguente:

SEPARAZIONE DELLE INFORMAZIONI DI STATO

FASE 1) LEGGE I DATI DI STATO DALLA PERIFERICA
FASE 2) OPERA L'AND LOGICO CON UNA MASCHERA (la maschera ha degli uni nelle posizioni da preservare, zeri altrove).

Uno scorrimento od operazione di posizionamento flag (per esempio ORA A) può sostituire la Fase 2 se il campo è un bit singolo ed occupa la posizione meno significativa, più significativa o successiva a quella più significativa. Queste posizioni sono spesso riservate all'informazione di stato più frequentemente impiegata.

COMBINAZIONE DELL'INFORMAZIONE DI CONTROLLO

La procedura di posizionamento o ripristino dei bit di controllo è la seguente:

FASE 1) CARICA L'INFORMAZIONE DI CONTROLLO PRIORITARIA
FASE 2) OPERA L'AND LOGICO CON UNA MASCHERA PER RIPRISTINO BIT (la maschera ha zeri nelle posizioni di bit da ripristinare ed uni altrove).
FASE 3) OPERA L'OR LOGICO CON UNA MASCHERA PER PORRE AD 1 DEI BIT (la maschera naturalmente ha uni in corrispondenza di bit da porre ad 1 e zeri altrove).
FASE 4) INVIA LA NUOVA INFORMAZIONE DI CONTROLLO ALLA PERIFERICA

Anche qui la procedura è più semplice se il campo è un bit singolo ed occupa una posizione alla fine della parola.

Questo è illustrato dagli esempi seguenti:

1) Un campo di tre bit nelle posizioni bit da 2 a 4 della porta d'ingresso 1 è un fattore di scala. Si posizioni questo fattore nell'Accumulatore.

LEGGE I DATI DI STATO DALLA PORTA D'INGRESSO

```
        IN      1           ;LEGGE I DATI DI STATO
```

MASCHERA OFF IL FATTORE DI SCALA E FALLO SCORRERE

```
        ANI     00011100B   ;MASCHERA IL FATTORE DI SCALA
        RRC                 ;SCORRIMENTO DOPPIO A DESTRA PER LA
                            ;NORMALIZZAZIONE
        RRC
```

2) I bit 2 e 3 nell'Accumulatore formano un campo di 2 bit che viene posizionato nelle posizioni 5 e 6 della porta d'uscita 4. La locazione di memoria 40 contiene i bit da 0 a 4 ed il bit 7 della parola da inviare alla porta di uscita 4 cioè (40) è l'attuale informazione di controllo.

MUOVE I DATI ALLE POSIZIONI DI CAMPO

```
        RLC                     ;FAI SCORRERE I DATI ALLE POSIZIONI 5 E 6
        RLC
        RLC
        ANI     01100000B       ;AZZERA GLI ALTRI BI
        MOV     B,A
```

COMBINA NUOVE POSIZIONI DI CAMPO CON DATI VECCHI

```
        LDA     40H             ;ACCETTA DATI VECCHI
        ANI     10011111B       ;AZZERA LE POSIZIONI DI CAMPO
        ORA     B               ;SOMMA DATI NUOVI
        OUT     4
```

È necessaria una copia dell'informazione di controllo poichè la CPU non può leggere una porta di uscita.

DOCUMENTAZIONE DI STATO E TRASFERIMENTI DI CONTROLLO

La documentazione è un serio problema nella trattazione delle informazioni di controllo e di stato. Il significato degli ingressi di stato od uscite di controllo è raramente ovvio. Il programmatore dovrebbe indicare chiaramente gli scopi delle operazioni di ingresso ed uscita nei commenti, per esempio «CONTROLLA SE IL LETTORE È ACCESO», «SCEGLIE PARITÀ PARI» oppure «ATTIVA IL CONTATORE DELLA VELOCITÀ DI BIT». Le istruzioni logiche e di scorrimento in caso contrario saranno molto difficili da ricordare, capire e collaudare (bedug).

ESEMPI

Una Tastiera non Codificata

Scopo: Riconoscimento della chiusura di un tasto da una tastiera non codificata 3 x 3 e posizionamento del numero di detto tasto nell'Accumulatore.

Una tastiera è un insieme di interruttori (Vedere Figura 11-16). Piccoli numeri di tasti sono facili da maneggiare se ogni tasto è collegato separatamente ad un bit oppure da una porta d'ingresso. La realizzazione dell'interfaccia di una tastiera è quindi analoga a quella di un gruppo di interruttori.

TASTIERA A MATRICE

Le tastiere con più di otto tasti richiedono più di una porta d'ingresso e quindi operazioni multibyte. Questo è particolarmente scomodo se i tasti sono separati logicamente cioè l'utente troverà particolari difficoltà nell'impiego del computer e del terminale a tastiera. Il numero di linee d'ingresso richiesto può essere ridotto connettendo il tasto a matrice come mostrato in Fig. 11-17. Ora ogni tasto rappresenta una potenziale connessione tra una riga ed una colonna. La tastiera ha matrice ha n + m linee esterne dove n è il numero di righe ed m il numero delle colonne. Questo porta ad n x m linee esterne se ogni tasto è separato. La Tabella 11-4 mostra il confronto fra alcune configurazioni tipiche.

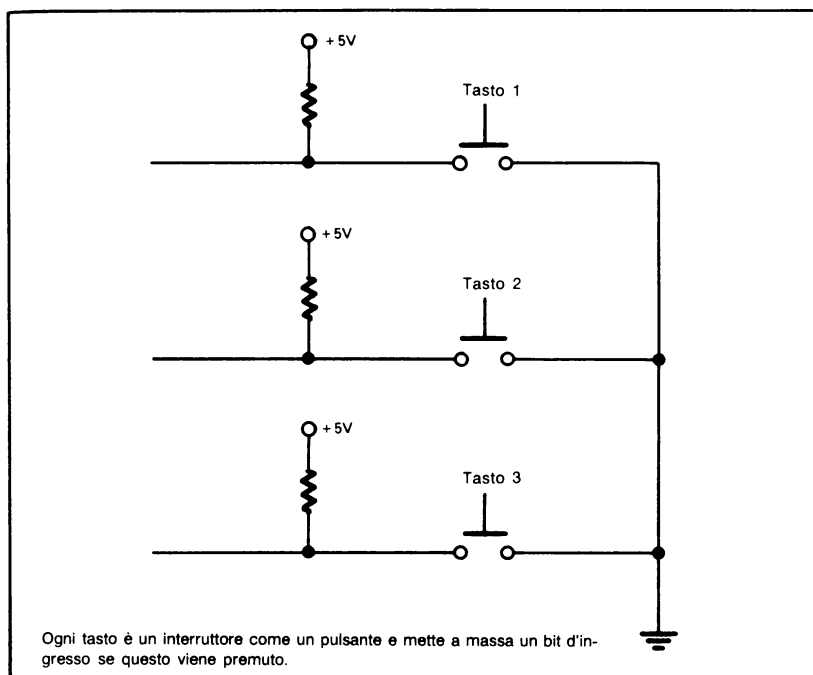



Figura 11-16
Una Piccola Tastiera

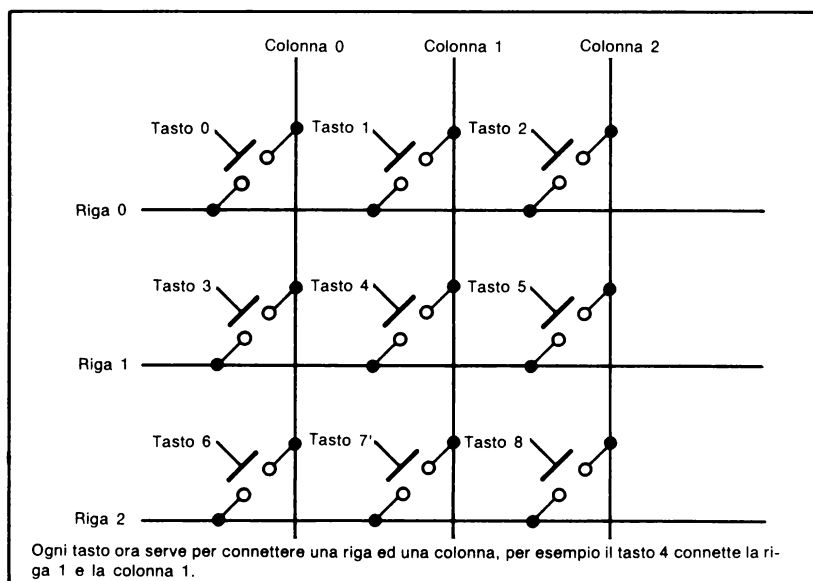



Figura 11-17
Una Tastiera A Matrice

Tabella 11-4
Confronto tra le connessioni indipendenti e le connessioni a matrice per le tastiere

Dimensioni Tastiera	Numero di linee con connessioni indipendenti	Numero di linee con la matrice di connessione
3 x 3	9	6
4 x 4	16	8
4 x 6	24	10
5 x 5	25	10
6 x 6	36	12
6 x 8	48	14
8 x 8	64	16

ESPLORAZIONE DELLA TASTIERA

La programmazione coinvolge problemi come la determinazione di quale tasto è stato premuto impiegando le linee esterne dalla matrice. La procedura usuale è una «esplorazione della tastiera». Viene collegata a massa la riga 0 e vengono esaminate le colonne. Se una di queste è a massa il tasto corrispondente è stato premuto originando la connessione riga-colonna. È possibile determinare quale tasto è stato premuto determinando quale colonna è a massa, cioè quale bit è '0' nella porta d'ingresso. Invece se nessuna colonna è a massa occorre procedere alla riga 1 e ripetere l'esplorazione delle colonne. Si può controllare per vedere se nessun tasto della tastiera è stato premuto collegando a massa tutte le righe contemporaneamente ed esaminando le colonne.

L'esplorazione della tastiera richiede che le righe siano collegate ad una porta d'uscita e le colonne ad una porta d'ingresso. La Figura 11-18 mostra tale disposizione. La CPU può collegare a massa una particolare riga posizionando uno '0' in un bit appropriato della porta di uscita e tutti uni negli altri bit. La CPU determina lo stato di una particolare colonna esaminando il bit corrispondenite della porta d'ingresso.

Compito 1: Attesa per la chiusura di un tasto.

La procedura è la seguente:

ATTESA PER LA CHIUSURA DI UN TASTO

1) Si colleghino a massa tutte le righe posizionando zeri in tutti i bit della porta d'uscita.
2) Si accettano gli ingressi di colonne leggendo la porta d'ingresso.
3) Ritorno alla Fase 1 se tutti gli ingressi di colonna sono uni.

Diagramma di flusso:

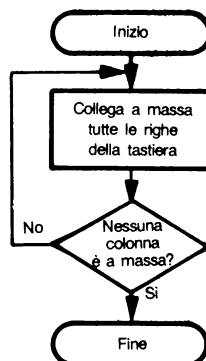
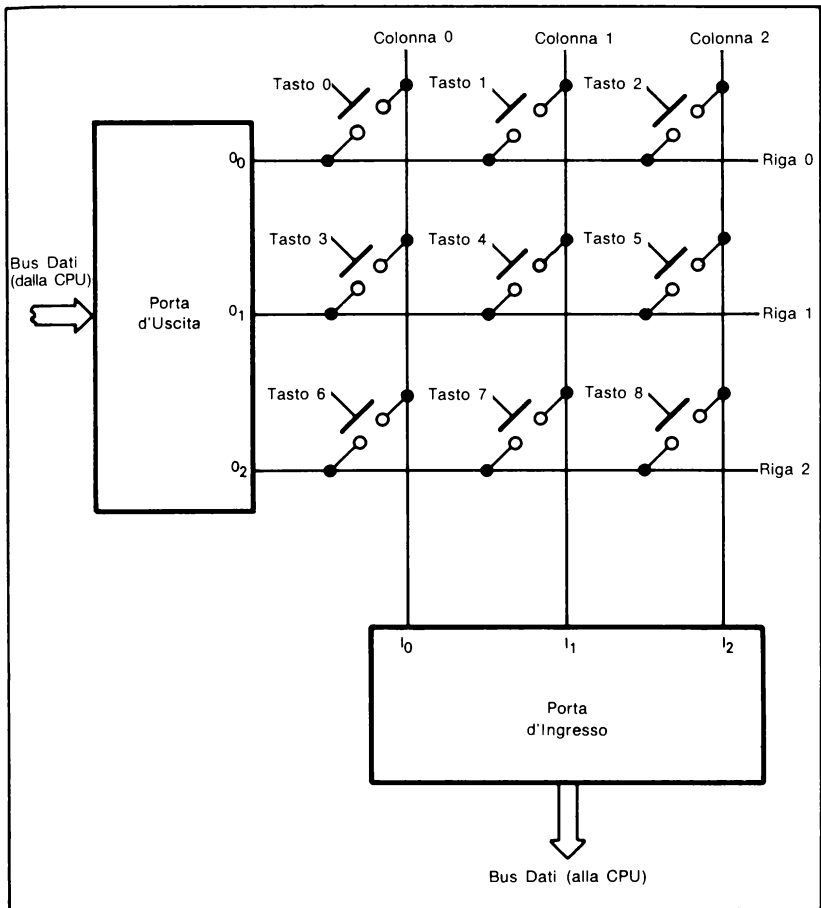



Figura 11-18
Disposizione I/O Per L'Esplorazione Di Una Tastiera

Programma Sorgente:

```
WAITK:  MVI     A,11111000B
        OUT     KBDOT       ;COLLEGAMENTO A TERRA DI TUTTE LE RIGHE
                            ;DELLA TASTIERA
        IN      KBDIN       ;ACCETTA DATI DALLA COLONNA DELLA TASTIERA
        ANI     00000111B   ;MASCHERA I BIT DI COLONNA
        CPI     00000111B   ;NESSUNA COLONNA È COLLEGATA A MASSA?
        JZ      WAITK       ;NO, PROSEGUE ANALISI DELLA TASTIERA
DONE:   JMP     DONE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00	WAITK:	MVI	A,11111000B	3E
01				F8
02		OUT	KBDOT	D3
03				KBDOT
04		IN	KBDIN	DB
05				KBDIN
06		ANI	00000111B	E6
07				07
08		CPI	00000111B	FE
09				07
0A		JZ	WAITK	CA
0B				00
0C				00
0D	DONE:	JMP	DONE	C3
0E				0D
0F				00

KBDOT è la porta d'uscita della tastiera, KBDIN è quella di ingresso.

Posizionando off tutti i bit di colonna si elimina qualunque problema che potrebbe essere causato dagli stati delle linee d'ingresso non utilizzate.

Si potrebbe generalizzare la routine assegnando dei nomi alle uscite e le maschere, cioè

```
ALLG    EQU    11111000B
OPEN    EQU    00000111B
```

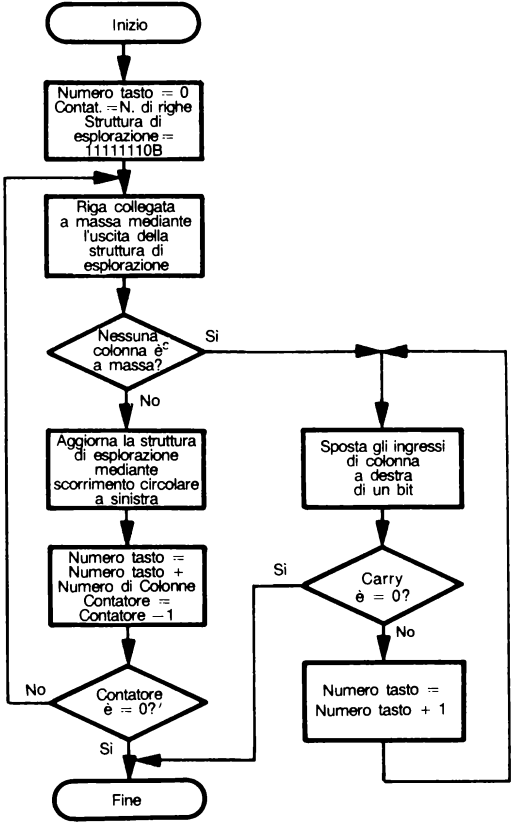
Questi nomi potrebbero essere poi impiegati nel programma effettivo; una diversa tastiera potrebbe richiedere solo un cambiamento nelle definizioni ed un riordinamento.

Compito 2: Identificazione della chiusura di un tasto posizionando il numero del tasto stesso nell'Accumulatore.

IDENTIFICAZIONE DI CHIUSURE DI TASTO

La procedura è la seguente:

1) Pone il numero di tasto a 0, il contatore uguale al numero delle righe e la struttura di uscita a tutti uni eccetto di uno '0' nel bit 0.
2) Pone a massa inviando la struttura di uscita alla porta di uscita della tastiera.
3) Aggiorna la struttura di uscita facendo scorrere il bit '0' a sinistra di una posizione.
4) Accetta gli ingressi di colonna leggendo la porta d'ingresso.
5) Se tutti gli ingressi di colonna sono '0' procede alla Fase 8.
6) Somma il numero di colonne al numero di tasto per ottenere la riga successiva.
7) Decrementa il contatore. Va alla fase 2 se tutte le righe non sono state esplorate, altrimenti va alla Fase 10.
8) Fa scorrere gli ingressi di colonna a destra di un bit.
9) Se Carry = 1 somma 1 al numero di tasto e ritorna alla Fase 8.
10) Fine del programma.

Diagramma di Flusso:

Programma Sorgente:

```
        MVI     B,0          ;NUMERO TASTO = 0
        MVI     C,11111110B  ;INIZIO ESPLORAZIONE DELLA STRUTTURA
                             ;METTE A MASSA LA RIGA 0
        MVI     D,3          ;CONTATORE = NUMERO DI RIGHE
FROW:   MOV     A,C
        OUT     KBDOT        ;ESPLORA UNA RIGA
        RLC                  ;AGGIORNA L'ESPLORAZIONE DELLA STRUTTURA
                             ;PER LA RIGA SUCCESSIVA
        MOV     C,A
        IN      KBDIN        ;ACCETTA GLI INGRESSI DI COLONNA
        ANI     00000111B    ;MASCHERA I BIT DI COLONNA
        CPI     00000111B    ;NESSUNA COLONNA È A MASSA?
        JNZ     FCOL         ;SÌ, RICERCA QUALE
        MOV     A,B          ;NUMERO TASTO = NUMERO TASTO +
                             ;NUMERO DI COLONNE
        ADI     3
        MOV     B,A
        DCR     D            ;TUTTE LE RICERCHE SONO STATE ESPLORATE?
        JNZ     FROW         ;NO, ESPLORA LA SUCCESSIVA
        JMP     DONE         ;SÌ, SALTA A DONE
FCOL:   RAR                  ;QUESTA COLONNA È A MASSA?
        JNC     DONE         ;SÌ, SALTA A DONE
        INR     B            ;NO, NUMERO TASTO = NUMERO TASTO + 1
        JMP     FCOL         ;ESAMINA LA COLONNA SUCCESSIVA
DONE:   JMP     DONE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		MVI	B,0	06
01				00
02		MVI	C,11111110B	0E
03				FE
04		MVI	D,3	16
05				03
06	FROW:	MOV	A,C	79
07		OUT	KBDOT	D3
08				KBDOT
09		RLC		07
0A		MOV	C,A	4F
0B		IN	KBDIN	DB
0C				KBDIN
0D		ANI	00000111B	E6
0E				07
0F		CPI	00000111B	FE
10				07
11		JNZ	FCOL	C2
12				1F
13				00
14		MOV	A,B	78
15		ADI	3	C6
16				03
17		MOV	B,A	47
18		DCR	D	15
19		JNZ	FROW	C2
1A				06
1B				00
1C		JMP	DONE	C3
1D				27
1E				00
1F	FCOL:	RAR		1F
20		JNC	DONE	D2
21				27
22				00
23		INR	B	04
24		JMP	FCOL	C3
25				1F
26				00
27	DONE:	JMP	DONE	C3
28				27
29				00

Ogni volta che viene eliminata una riga esplorata, occorre sommare il numero di colonne al numero di tasto in modo da passare dalla riga alla nuova (si verifichi sulla tastiera in Figura 11-18).

Questo programma può essere generalizzato assegnando nomi di parametri al numero di righe, numero di colonne alla struttura da mascherare per mezzo dell'impiego di pseudo-operazioni EQU.

In questo programma ci sono tre salti incondizionati. È possibile eliminarne due mediante un'appropriata ristrutturazione delle condizioni iniziali? Inoltre è possibile accorciare la procedura mediante la quale viene aggiunto 3 al Registro B?

Una Tastiera Codificata

Scopo: Attende la disponibilità di dati da una tastiera codificata che fornisce, assieme ai dati, un segnale di strobe. Posiziona i dati nell'Accumulatore.

TASTIERE CODIFICATE

Una tastiera codificata fornisce un unico codice di identificazione per ogni tasto. Essa è dotata di una parte elettronica interna che esegue la procedura di scansione ed identificazione descritta nell'esempio precedente. C'è un compromesso tra un software più semplice e più alti cosi della tastiera codificata contro un software più complesso e costi più bassi della tastiera codificata.

Le tastiere codificate possono impiegare matrici di diodi, codificatori TTL ovvero codificatori MOS. I codici di identificazione possono essere ASCII, EBCDIC o qualche altro codice. Spesso le PROM sono parte della circuiteria di codifica.

ROLLOVER

La circuiteria di codifica può fare qualcosa di più della trasformazione della chiusura di tasti in codice. Essa può anche eliminare il rimbalzo dei tasti e maneggiare «rollover», il problema che sorge quando più di un tasto viene premuto allo stesso istante. I metodi più comuni per trattare il rollover comprendono «rollover di 2 tasti» (2KRO) per cui 2 tasti (ma non di più), premuti allo stesso istante, vengono trattati come chiusure separate, ed il «rollover di n tasti» (NKRO) dove un numero qualsiasi di tasti premuti allo stesso istante, vengono risolti in chiusure separate.

La tastiera codificata fornisce anche un segnale strobe con ogni nuovo ingresso dati. Lo strobe indica che è presente un nuovo dato. La Figura 11-19 mostra la realizzazione dell'interfaccia tra una tastiera codificata ed un microprocessore 8080. Lo strobe della tastiera viene memorizzato in un flip-flop SR della porta I/O 8212 (si noti che lo strobe è attivo al livello alto ma l'uscita del flip-flop è attiva al livello basso). Una porta d'ingresso seriale è necessaria poichè il processore non è dotato di un modo diretto per la determinazione dello stato del flip-flop SR. L'uscita INT dall'8212 è attiva al livello basso ed essa è intesa a servire come un segnale di interrupt, sia direttamente al processore sia per mezzo di un'Unità di Controllo della Priorità di Interrupt 8214. La Figura 11-20 è uno schema della porta I/O 8212 e la Figura 11-21 descrive il suo impiego come porta d'ingresso. (Si ricordi che il Volume II di An Introduction To Microcomputers descrive in dettaglio la porta I/O 8212). La maggior parte delle schede di I/O impiegate nei sistemi a microcomputer contiene sia porte dati paralleli sia porte di stato seriali.

La porta seriale può essere anche un singolo flip flop D indirizzato dalla CPU in modo da determinare il suo stato.

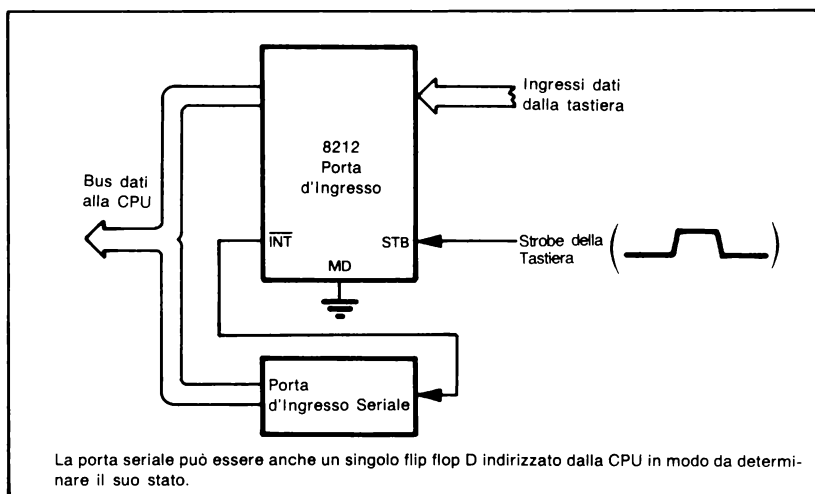
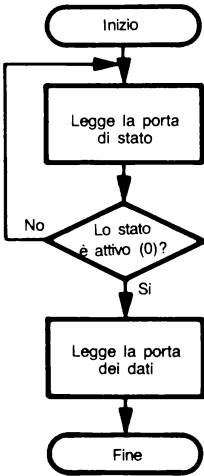



Figura 11-19
Realizzazione Dell'Interfaccia I/O Per Una Tastiera Codificata

Compito: Attende un segnale di strobe attivo al livello basso alla porta di stato seriale e quindi posiziona nell'Accumulatore i dati provenienti dalla porta parallela.
Si noti che la lettura di dati dalla porta parallela azzera il flip-flop SR (questa circuiteria fa parte dell'8212). Si assumerà che la porta di stato seriale sia connessa al bit 0 del Bus Dati.

Diagramma di Flusso:

STB	MD	($\overline{DS}_1$-DS_2)	Uscita dati Uguale
0	0	0	Tri state
1	0	0	Tri state
0	1	0	Latch Dati
1	1	0	Latch Dati
0	0	1	Latch Dati
1	0	1	Ingresso Dati
0	1	1	Ingresso Dati
1	1	1	Ingresso Dati

$\overline{CLR}$ = Ripristina il Latch Dati
Set del Flip-Flop SR
(Non ha effetti sul Buffer d'Uscita)

CLR	($\overline{DS}_1$-DS_2)	STB	* SR	INT
0	0	0	1	1
0	1	0	1	0
1	1	⌊	0	0
1	1	0	1	0
1	0	0	1	1
1	1	⌊	1	0

* Flip-Flop SR Interno

Figura 11-20
La Porta I/O 8212

Figura 11-20. La Porta I/O 8212 (seguito)

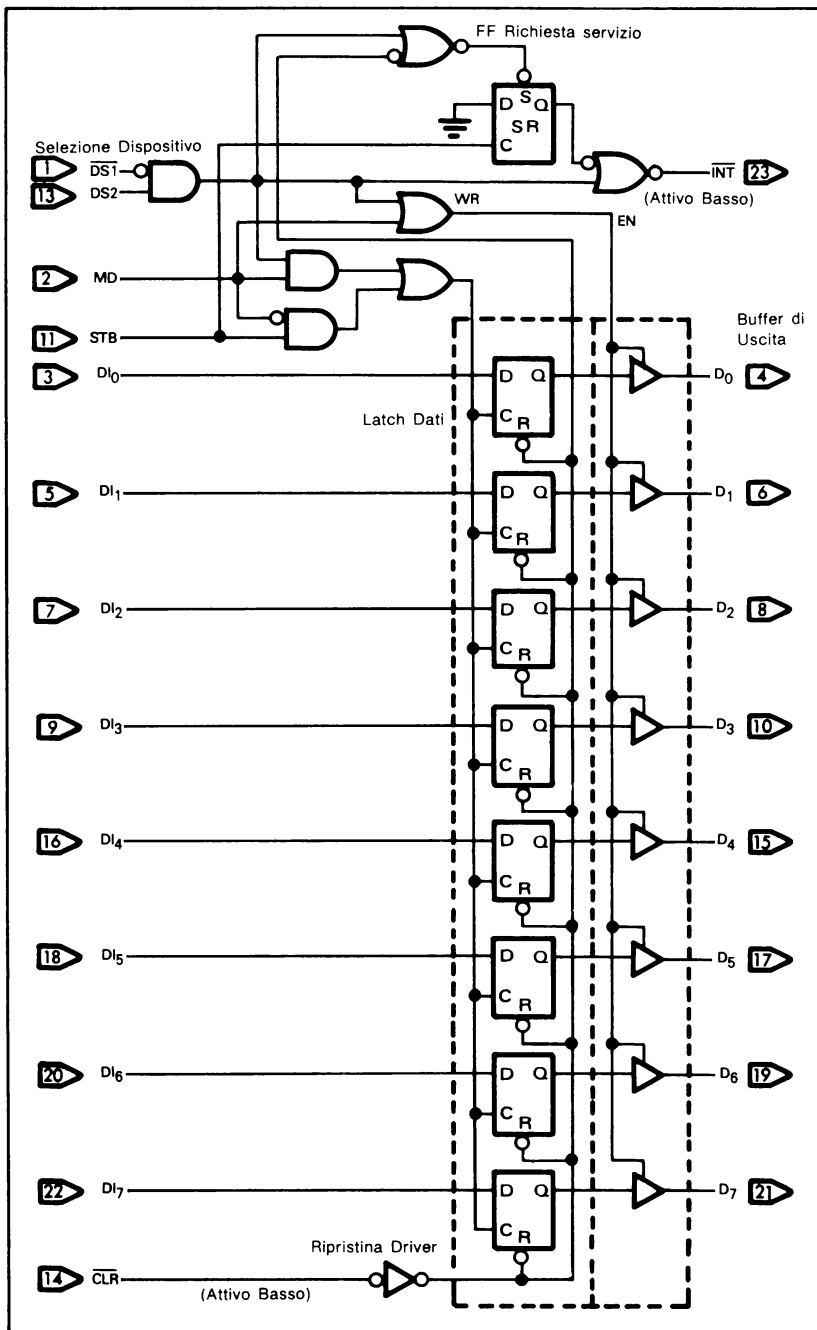
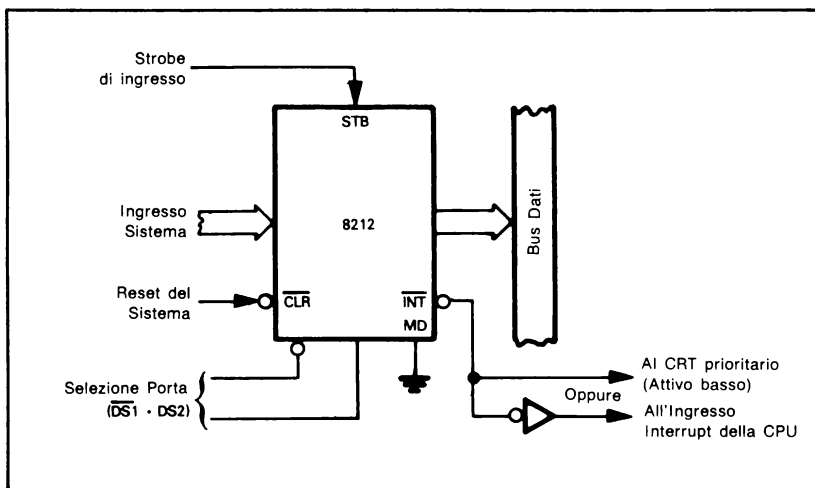



Figura 11-21
La Porta I/O 8212 Come Porta D'Ingresso Di Interrupt

L'impiego di un 8212, mostrato in Figura 11-21, è quello di una porta d'ingresso del sistema che accetta uno strobe dalla sorgente d'ingresso del sistema che di volta in volta azzera il flip-flop di richiesta servizio e gli interrupt del processore. Il processore poi esegue una routine di servizio, identifica la porta ed origina la selezione del dispositivo logico corretto — abilitando i dati d'ingresso del sistema verso il Bus Dati.

Programma Sorgente:

```
SRCHS:  IN      SPORT   ;LEGGI LA PORTA DI STATO
        RAR             ;C'È UN NUOVO DATO DALLA TASTIERA?
        JC      SRCHS   ;NO, PROSEGUI RICERCA
        IN      DPORT   ;SÌ, PRELEVA DATO
HERE:   JMP     HERE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)	Contenuti di Mem. (Esadec.)
00	SRCHS: IN SPORT	DB
01		SPORT
02	RAR	1F
03	JC SRCHS	DA
04		00
05		00
06	IN DPORT	DB
07		DPORT
08	HERE: JMP HERE	C3
09		08
0A		00

Si ricordi che il bit di stato è attivo basso, cioè uno «0» logico indica la presenza di nuovi dati.

Un Convertitore Digitale-Analogico

Scopo: Invia dati ad un convertitore digitale-analogico ad 8 bit che richiede un impulso di LOAD per iniziare il processo di conversione.

I convertitori digitale-analogico producono i segnali continui richiesti da motori, solenoidi, relé, attuatori ed altri dispositivi di uscita elettrici e meccanici. Convertitori tipici consistono di switch e scale di resistori con appropriati valori di resistenza. Tipicamente si deve fornire un potenziale di riferimento e qualche altra circuiteria digitale ed analogica sebbene comincino ad essere disponibili unità complete a basso costo (Si veda per esempio: A. Mrozowski «Analog Output Chips Shrink A-D Conversion Software» Electronics, Giugno 23, 1977, pp 130-133).

La Figura 11-22a descrive il convertitore D/A ad 8 bit Signetics NE5018, un esempio di dispositivo progettato per essere compatibile con il microprocessore comprendendo un latch per i dati di ingresso paralleli ad 8 bit sullo stesso chip. Un livello basso dell'ingresso LE (abilitazione latch) passa l'informazione del Bus Dati nei latch, dove resta dopo che LE passa al livello alto. La Figura 11-22b illustra la connessione del dispositivo ad un sistema 8080.

La Figura 11-23 mostra un'interfaccia per un convertitore ad 8 bit che impiega due porte di uscita, una per i dati del convertitore ed una per l'impulso di caricamento.

La porta di uscita dati non è necessaria se il convertitore ha un registro o latch e se la logica di selezione della porta può essere applicata direttamente all'ingresso strobe dei dati del convertitore (come sul Signetics NE5018).

Nelle applicazioni dove 8 bit non sono sufficienti è possibile impiegare dei convertitori da 10 a 16 bit. Una porta logica addizionale è richiesta per passare tutti i bit dati; alcuni convertitori forniscono una parte di questa logica.

Si noti che la porta 8212 è latched dalla logica di selezione del dispositivo nel modo di uscita (Vedere Figura 11-24). I dati perciò possono permanere durante e dopo la conversione. Il convertitore richiede tipicamente solo pochi microsecondi per produrre l'uscita analogica richiesta.

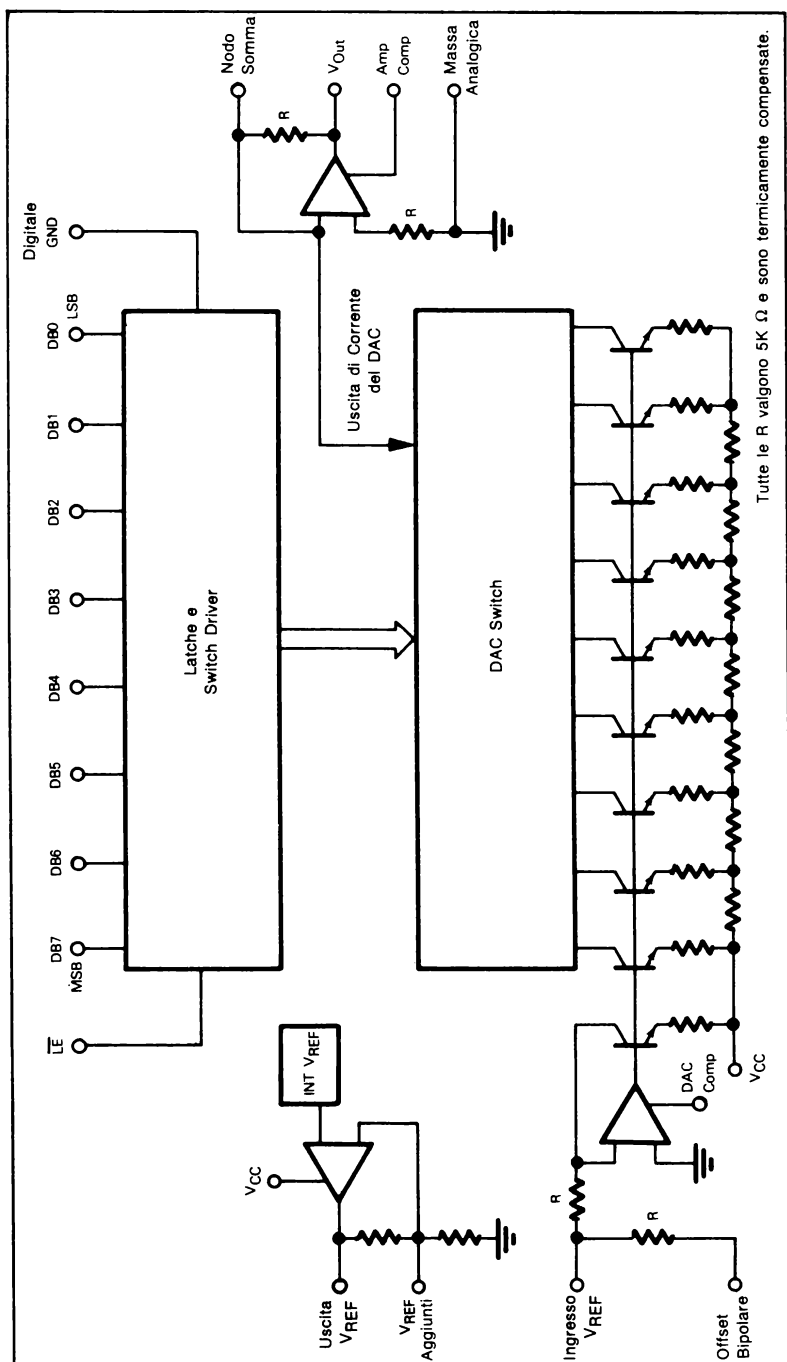



Figura 11-22a.
Convertitore D/A Signetics NE5018

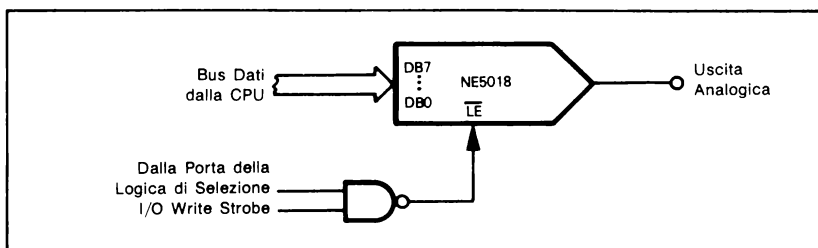



Figura 11-22b.
Connessione Dell'NE5018 Al Sistema 8080

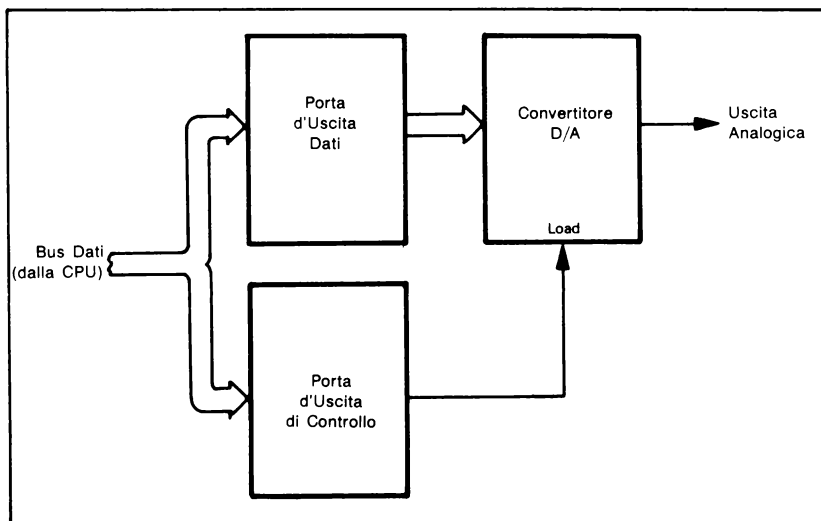



Figura 11-23
Interfaccia Per Un Convertitore Analogico-Digitale Ad 8-Bit

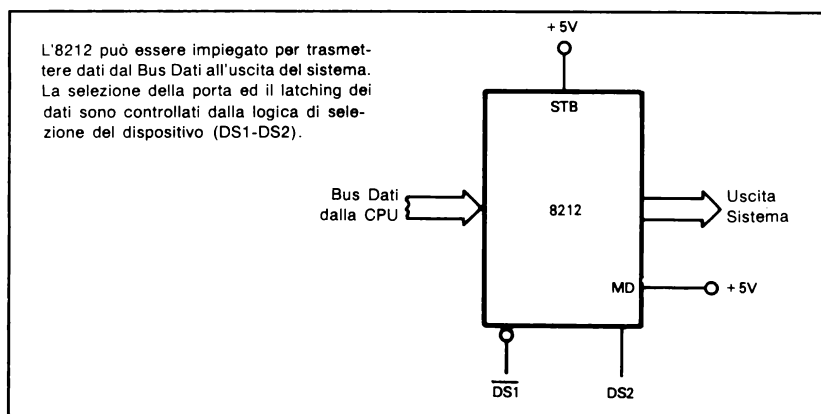
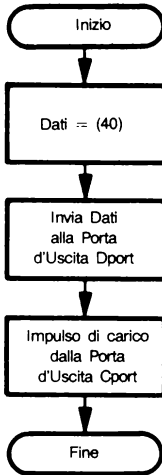



Figura 11-24
Porta di I/O 8212 Considerata Come Porta D'Uscita Latching

Compito: Invia i dati della locazione di memoria 40 al convertitore. Invia anche l'impulso che caricherà il convertitore.

Diagramma di Flusso:

Programma Sorgente:

(L'ingresso LOAD sul DAC è assunto connesso alla linea 0 del Bus Dati).

```
        LDA     40H     ;ACCETTA DATI
        OUT     DPORT   ;INVIA DATI AL DAC
        MVI     A,1     ;IMPULSO DI CARICAMENTO DEL DAC
        OUT     CPORT   ;CARICAMENTO DAC: INGRESSO ALTO
        SUB     A
        OUT     CPORT   ;CARICAMENTO DAC: INGRESSO BASSO
HERE:   JMP     HERE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)	Contenuti di Mem. (Esadec.)
00	LDA 40H	3A
01		40
02		00
03	OUT DPORT	D3
04		DPORT
05	MVI A,1	3E
06		01
07	OUT CPORT	D3
08		CPORT
09	SUB A	97
0A	OUT CPORT	D3
0B		CPORT
0C	HERE: JMP HERE	C3
0D		0C
0E		00

Se gli altri bit di CPORT sono impegnati, il segnale LOAD può essere inviato operando l'OR di una copia dei vecchi dati di controllo con una maschera opportuna. Tale maschera avrebbe un bit '1' nella posizione LOAD e zeri in tutte le altre. Per esempio se i vecchi dati di controllo sono nella locazione di memoria 40H si potrebbe applicare la seguente sequenza di istruzioni:

```
LDA   40H          ;ACCETTA VECCHI DATI DI CONTROLLO
ORI   00000001B    ;COMMUTA ON IL BIT 0"
OUT   CPORT
```

L'impulso potrebbe poi essere completato operando l'AND con una maschera che ha un bit '0' nella posizione LOAD ed uni in tutte le altre. Per esempio se i dati di controllo sono nell'Accumulatore,

```
ANI   11111110B
OUT   CPORT
```

porterebbero al livello basso la linea LOAD.

L'impulso normalmente deve avere una durata non inferiore ad un minimo. Questo minimo potrebbe essere di diversi cicli di clock. Quando LOAD è basso l'uscita analogica non varia.

Il segnale $\overline{INT}$ dal dispositivo 8212 può servire come «BYTE OUT» oppure un altro segnale di controllo, poichè lo attiva la logica di selezione del dispositivo. Si noti, comunque, che l'impulso INT è uno strobe abbastanza breve perché dura soltanto un ciclo di clock.

Un Convertitore Analogico-Digitale

Scopo: Prende i dati da un convertitore analogico-digitale ad 8 bit che richiede un impulso STAR CONVERSION per iniziare il processo di conversione ed è dotato di una linea BUSY per indicare il completamento del processo.

I convertitori analogico-digitale trattano segnali continui prodotti da vari tipi di sensori e trasduttori. Il convertitore produce il segnale d'uscita digitale che richiede il computer.

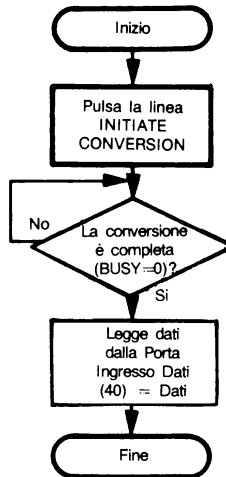
Una forma di convertitore analogico-digitale è il dispositivo ad approssimazioni successive che opera un confronto diretto di 1 bit durante ogni ciclo di clock. Tali convertitori sono veloci ma hanno una bassa immunità al rumore. I convertitori ad integrazione a doppia pendenza sono un'altra forma di convertitori analogico-digitale. Questi dispositivi sono più lenti ma più immuni al rumore. Altre tecniche, come la tecnica di bilanciamento della carica incrementale, vengono spesso impiegate.

I convertitori analogico-digitale normalmente richiedono qualche circuiteria esterna analogica e digitale. Unità complete cominciano ad essere disponibili a basso costo.

La Figura 11-25 mostra il Convertitore A/D 8703 ad 8 bit della Teledyne Semiconductor. Il dispositivo contiene un risultato latch ed uscite dati tri-state. Un impulso su INITIATE CONVERSION inizia la conversione dell'ingresso analogico; dopo circa due millisecondi il risultato andrà ai latch di uscita e l'uscita BUSY lo indicherà passando al livello basso. I dati sono letti dai latch applicando uno '0' all'ingresso ENABLE.

La Figura 11-26 mostra l'interfaccia per un convertitore ad 8 bit. (Vedere anche D. Guzeman, «Marry Your µP To Monolithic A/Ds», Electronic Design, 18 Gennaio 1977, pp. 82-86). Le varie porte possono essere necessarie se il convertitore (come il Teledyne 8703) ha uscite tri-state e circuiteria di abilitazione separata per le varie funzioni. Si noti che, accanto alle porte di ingresso dati e di stato è necessaria una porta di controllo di uscita per fornire l'impulso di START CONVERSION. Il segnale BUSY potrebbe anche chiedere i dati del convertitore nella porta dati d'ingresso.

Compito: Inizia il processo di conversione, attende che BUSY sia al livello alto e quindi legge i dati e li immagazzina nella locazione di memoria 40.

Diagramma di Flusso:

Programma Sorgente:

(Sia le porte di controllo che quelle di stato sono assunte connesse al bit 0 del Bus Dati)

Le operazioni logiche OR ed AND potrebbero essere impiegate per pulsare la linea INITIATE CONVERSION.

Una routine di ritardo di opportuna lunghezza (2 ms per l'8703) potrebbe sostituire l'esame del segnale BUSY.

Il segnale BUSY è assunto essere '1' quando è in corso la conversione.

```
        MVI   A,1     ;IMPULSO INITIATE CONVERSION
        OUT   CPORT   ;INITIATE CONVERSION = 1
        SUB   A
        OUT   CPORT   ;INITIATE CONVERSION = 0
WTBSY:  IN    SPORT   ;LA CONVERSIONE È COMPLETA (BUSY = 0)?
        RAR
        JC    WTBSY   ;NO, ATTENDI
        IN    DPORT   ;SÌ, ACCETTA DATI
        STA   40H
HERE:   JMP   HERE
```

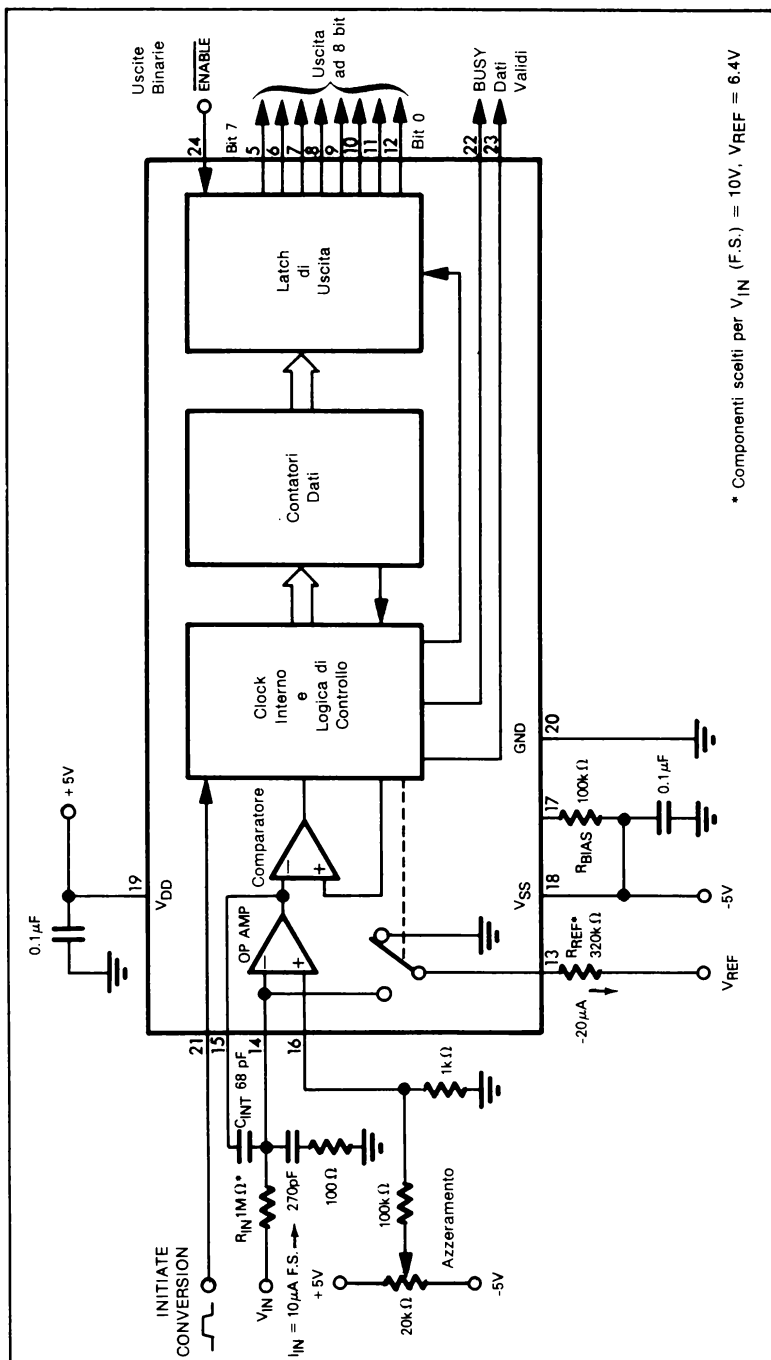



Figura 11-25
Convertitore A/D Teledyne 8703

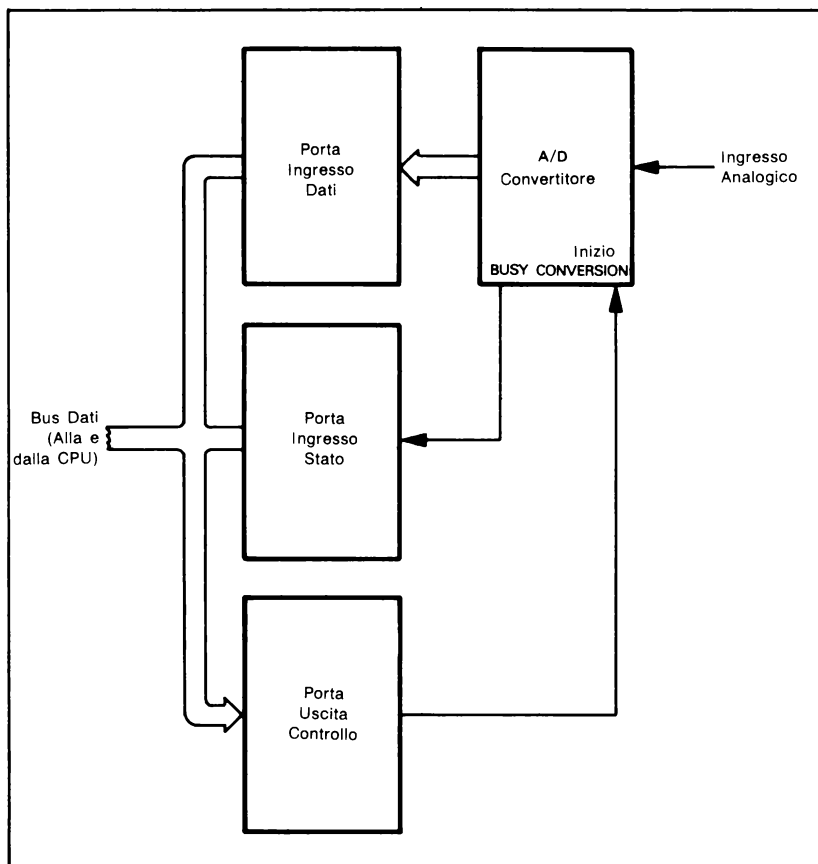



Figura 11-26
Interfaccia Per Un Convertitore Analogico-Digitale Ad 8 Bit

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		MVI	A,1	3E
01				01
02		OUT	CPORT	D3
03				CPORT
04		SUB	A	97
05		OUT	CPORT	D3
06				CPORT
07	WTBSY:	IN	SPORT	DB
08				SPORT
09		RAR		1F
0A		JC	WTBSY	DA
0B				07
0C				00
0D		IN	DPORT	DB
0E				DPORT
0F		STA	40H	32
10				40
11				00
12	HERE:	JMP	HERE	C3
13				12
14				00

Una Telescrivente (TTY)

Scopo: Trasferire dati a e da una telescrivente seriale standard a 10 caratteri al secondo.

INTERFACCIA DI TELESCRIVENTE

La telescrivente standard (Modello Teletype ASR-33) trasferisce dati in modo asincrono seriale. La procedura è la seguente:

1) La linea è normalmente nello stato MARK oppure '1'.
2) Un bit di Inizio (uno Spazio od un bit '0') precede ogni carattere.
3) Il carattere è normalmente ASCII a 7 bit con precedenza di trasmissione al bit meno significativo.
4) Il bit più significativo è quello di parità, che può essere pari, dispari, oppure fissa a '0' oppure '1'.
5) Due bit di Arresto seguono ogni carattere ('1' logico).

FORMATO DEI CARATTERI PER LA TTY STANDARD

La Figura 11-27 mostra il formato. Si noti che ogni carattere in realtà richiede la trasmissione di undici bit di cui solo sette contengono l'informazione. Poichè la velocità massima dei dati è di dieci caratteri al secondo la velocità di bit e 10 x 11 ovvero 110 Baud. Ogni bit perciò ha una larghezza di 1/110 di secondo ovvero 9,1 millisecondi. Questa larghezza è una media; la telescrivente non la conserva per un livello di precisione comunque elevato.

Per una telescrivente sono necessarie le seguenti procedure per comunicare correttamente col computer:

RICEZIONE (Diagramma di flusso in Figura 11-28).

FASE 1) Ricerca un bit di Partenza cioè uno '0' logico sulla linea dati.
FASE 2) Centra la ricezione attendendo metà tempo di bit cioè 4,55 millisecondi.
FASE 3) Prende i bit dati, attende un tempo di bit prima di ognuno.
Fa entrare i bit dati in una parola mediante un primo spostamento del bit Carry e quindi scorrendo circolarmente i dati con il Carry.
FASE 4) Genera la Parità e la controlla dopo la ricezione. Se non sono uguali indica un «errore di Parità».
FASE 5) Prende i bit di Arresto (attendendo un tempo di bit tra gli ingressi). Se questi non sono corretti (cioè se entrambi i bit di Arresto non sono '1') indica un «errore di struttura».

MODO RICEZIONE DELLA TTY

Compito: Prende dati da una telescrivente attraverso una porta di ingresso seriale e posiziona i dati nella locazione di memoria 60. Per la procedura si veda la Figura 11-28.

V PAG 11-61

V PAG 11-62

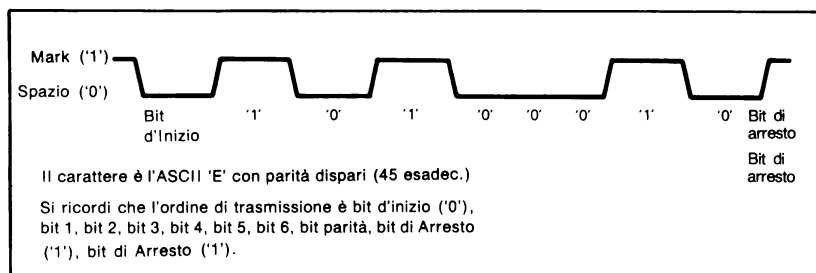



Figura 11-27
Formato Dati Di Telescrivente

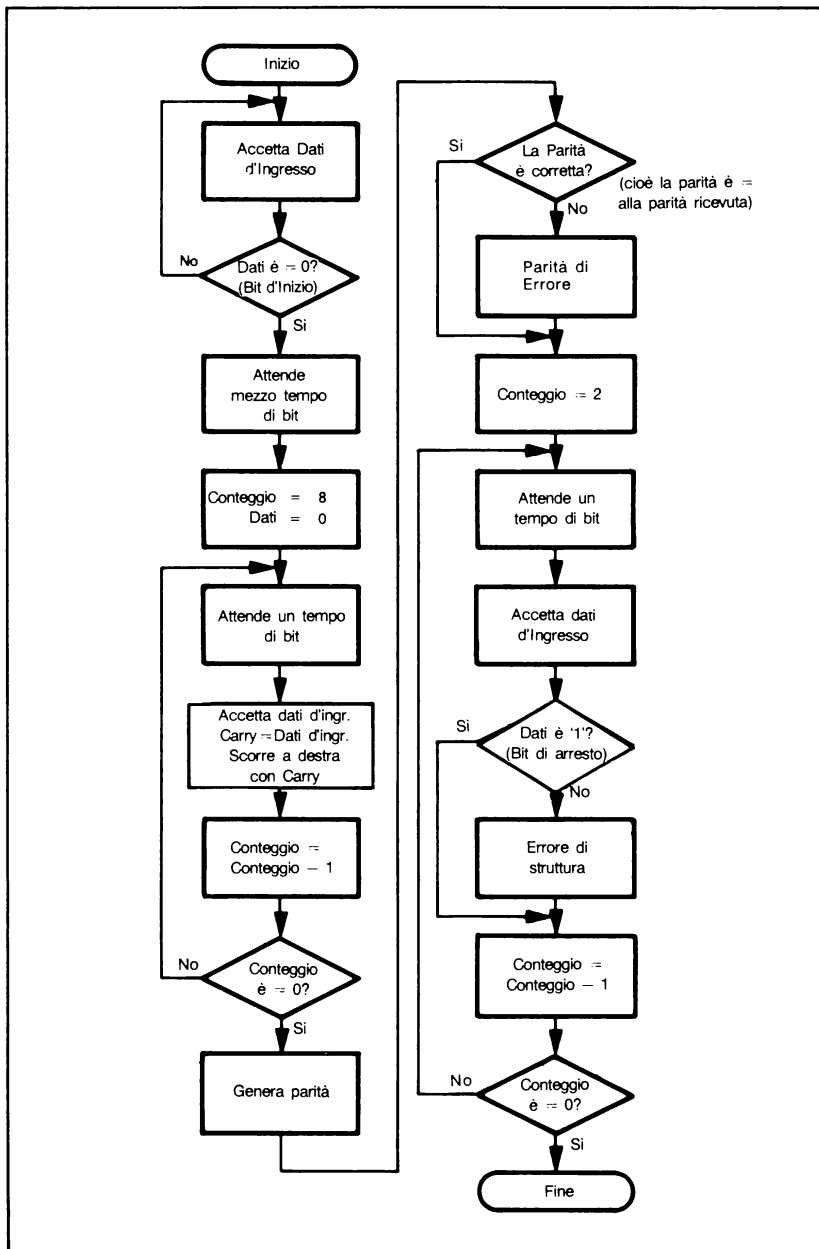



Figura 11-28
Diagramma Di Flusso Per La Procedura Di Ricezione

Programma Sorgente:

PROGRAMMA DI RICEZIONE TTY

(Si assume che la porta seriale sia connessa al bit 0 del Bus Dati e le subroutine PRERR e FRERR maneggiano parità ed errori di struttura rispettivamente). La parità sia dispari.

```
;
;ATTENDE PER IL BIT DI INIZIO
;
;WTSTB:  IN      RPORT       ;ACCETTA DATI
         RAR                 ;DATI È UN BIT D'INIZIO ('0')?
         JC      WTSTB       ;NO, ATTENDI
;
;CENTRO RICEZIONE E MARK DELLA PAROLA DATI
;
         CALL    DHALF       ;ATTENDE 1/2 TEMPO DI BIT
         MVI     D,10000000B ;MARK DELLA PAROLA DATI
;
;ACCETTA DATI SERIALI E POSIZIONALI NELLA PAROLA DATI
;
RCVB:    CALL    DFULL       ;ATTENDE 1 TEMPO DI BIT
         IN      RPORT       ;ACCETTA DATI SERIALI
         RAR
         MOV     D,A
         RAR                 ;FA SCORRERE I DATI SERIALI NELLA PAROLA
         MOV     A,D         ;MARK HA ATTRAVERSATO LA PAROLA?
         JNC     RCVB        ;NO, CONTINUA ACCETTAZIONE BIT DEL CARATTERE
;
;CONTROLLA PARITÀ E CONSERVA DATI
;
         ANA     A           ;PONE 1 IL BIT PARITÀ
         JPE     PRERR       ;ERRORE DI PARITÀ SE LA PARITÀ RICEVUTA È PARI
         STA     60H
;
;ACCETTA E CONTROLLA I BIT DI ARRESTO
;
         MVI     E,2         ;2 BIT DI ARRESTO
RCVS:    CALL    DFULL       ;ATTENDE 1 TEMPO DI BIT
         IN      RPORT       ;ACCETTA DATI SERIALI
         RAR                 ;DATI È UN '1'?
         JNC     FRERR       ;NO, ERRORE DI STRUTTURA
         DCR     E
         JNZ     RCVS
HERE:    JMP     HERE
;
;ROUTINE DI RITARDO
;
DHALF:   MVI     B,4         ;1/2 TEMPO DI BIT
         JMP     DLY
DFULL:   MVI     B,8         ;1 TEMPO DI BIT
DLY      MVI     C,150       ;RITARDO DI 1/8 DI TEMPO DI BIT
         DCR     C
         JNZ     DLY1
         DCR     B
         JNZ     DLY
         RET
```

Un semplice metodo di controllo per vedere quando è stata ricevuta una parola completa è di posizionare un '1' nel bit più significativo della parola dati e zeri negli altri. Quando compare questo '1' ulteriore nel Carry, un'intera parola di dati è stata ricevuta e fatta scorrere (a destra).

Il Puntatore dello Stack deve essere inizializzato poichè le routine di ritardo vengono richiamate come subroutine.

La routine di ritardo può essere facilmente fornita di un ritardo pari a metà tempo di bit ovvero di un intero tempo di bit con due punti di ingresso separati.

Potrebbe anche essere impiegata la linea di ingresso seriale (SID) dell'8085. L'istruzione RIM (Leggi la Maschera di Interrupt) carica l'Accumulatore con la linea d'ingresso seriale nel bit 7. Così i cambiamenti richiesti sono:

```
;
;ATTENDE I BIT DI PARTENZA
;
WTSTB:  RIM             ;ACCETTA DATI SERIALI
        RAL             ;SID È UN BIT DI PARTENZA ('0')?
        JC      WTSTB   ;NO, ATTENDI
;
;ACCETTA DATI SERIALI E LI POSIZIONA NELLA PAROLA DATI
;
RCVB    CALL    DFULL   ;ATTENDE 1 TEMPO DI BIT
        RIM             ;ACCETTA DATI SERIALI
        RAL
```

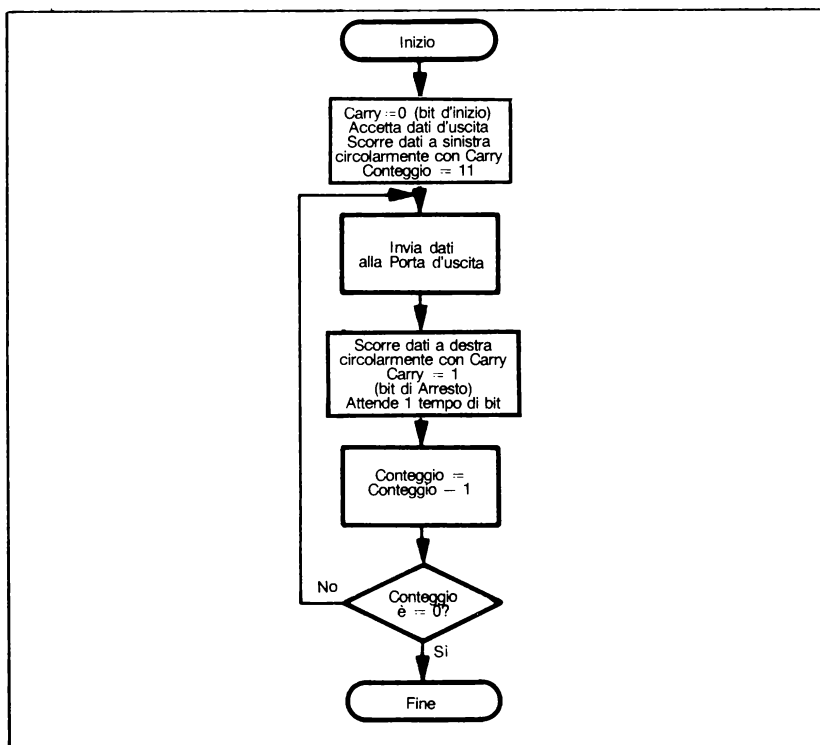



Figura 11-29
Diagramma Di Flusso Per La Procedura Di Trasmissione

TRASMISSIONE (Diagramma di Flusso in Figura 11-29)

FASE 1) Trasmette un bit di Inizio, cioè uno '0' logico.
FASE 2) Trasmette i sette bit dati iniziando col bit meno significativo.
FASE 3) Genera e trasmette il bit Parità.
FASE 4) Trasmette due bit di Arresto, cioè uni logici.

La routine di trasmissione deve aspettare un tempo di bit tra ogni operazione.

UART

Queste procedure sono sufficientemente comuni e complesse per meritare un dispositivo LSI speciale, l'UART ovvero il Ricevitore/Trasmettitore Asincrono Universale. (Per una discussione sugli UART vedere P. Rony et al. «The Bugbook II a» E and L Instruments Inc., 61 First St., Derby Conn. 06418, oppure D.G. Larsen et al., «INWAS: Interfacing with Asynchronous Serial Mode», IEEE Transactione on Industrial Electronics and Control Instrumentation, Febbraio 1977 pp 2-12). L'UART eseguirà la procedura di ricezione e fornirà i dati in forma parallela ed un segnale DATA READY. Esso accetterà anche dati in forma parallela, eseguirà una procedura di trasmissione e fornirà un segnale PERIPHERAL READY quando può trattare più dati. Gli UART possono avere molte altre caratteristiche tra cui:

1) Capacità di trattare varie lunghezze di carattere (normalmente da cinque ad otto bit), scelta parità e numero di bit di Arresto (normalmente 1, 1,5 e 2).
2) Indicatori per errori di struttura, errori di parità ed «errori di overrun» (difetto nella lettura di un carattere prima che un altro sia ricevuto).
3) Compatibilità con RS-232 (è possibile trovare delle descrizioni introduttive dell'RS-232 in F.W. Etcheverry «Binary Serial Interfaces», EDN, 20 aprile 1976, pp. 40-43 ed in G. Pickles «Who's Afraid of RS-232?», Kilobaud, Maggio 1977 pp. 50-54), cioè un segnale d'uscita REQUEST TO SEND (RTS) per indicare la presenza di dati nella fornitura di comunicazioni ed un segnale d'ingresso CLEAR-TO-SEND (CTS) per indicare, in risposta ad RTS la non disponibilità della fornitura di comunicazioni. Può esistere la necessità di altri segnali dell'RS-232 come RECEIVED SIGNAL QUALITY, DATA SET READY oppure DATA TERMINAL READY.
4) Uscite tri-state e compatibilità di controllo con microprocessore.
5) Scelte di clock che permettano all'UART di campionare i dati entranti diverse volte per rivelare bit di partenza falsi ed altri errori.
6) Possibilità di interrupt e controlli.

PROGRAMMA DI TRASMISSIONE DELLA TTY

Gli UART agiscono come quattro porte parallele — una porta dati d'ingresso, una porta dati d'uscita, una porta di stato d'ingresso ed una porta di controllo d'uscita. I bit di stato comprendono indicatori di errore come i flag READY. I bit di controllo selezionano varie scelte. Gli UART sono non dispendiosi (da $5 a $50 in dipendenza delle caratteristiche) e facile da usare.

Compito: Invia dati dalla locazione di memoria 60 alla telescrivente attraverso una porta d'uscita seriale.

Per la procedura si veda la Figura 11-29.

Programma Sorgente:

(Si assume che la porta seriale sia connessa al bit 0 del Bus Dati e la parità sia parte dei dati).

ACCETTA DATI ED AZZERA IL BIT D'INIZIO

```
        LDA     40H     ;ACCETTA DATI
        ADD     A       ;AZZERA IL BIT D'INIZIO
        MVI     D,11    ;CONTEGGIO = 11 BIT
```

TRASMETTE UN BIT ED AGGIORNA DATI

```
TBIT    OUT     TPORT   ;TRASMETTE UN BIT
        RAR             ;AGGIORNA PER IL BIT SUCCESSIVO
        STC             ;BIT DI ARRESTO = 1
```

RITARDO DI 9,1 MS

```
        MVI     B,8     ;ATTRAVERSAMENTO 8 VOLTE
DLY:    MVI     C,150   ;RITARDO DI 1/8 DEL TEMPO DI BIT
DLY1:   DCR     C
        JNZ     DLY1
        DCR     B
        JNZ     DLY
```

CONTEGGIO BIT

```
        DCR     D       ;CONTO ALLA ROVESCIA DI 11 BIT
        JNZ     TBIT
DONE:   JMP     DONE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		LDA	40H	3A
01				40
02				00
03		ADD	A	87
04		MVI	D,11	16
05				0B
06	TBIT:	OUT	TPORT	D3
07				TPORT
08		RAR		1F
09		STC		37
0A		MVI	B,8	06
0B				08
0C	DLY:	MVI	C,150	0E
0D				96
0E	DLY1:	DCR	C	0D
0F		JNZ	DLY1	C2
10				0E
11				00
12		DCR	B	05
13		JNZ	DLY	C2
14				0C
15				00
16		DCR	D	15
17		JNZ	TBIT	C2
18				06
19				00
1A	DONE:	JMP	DONE	C3
1B				1A
1C				00

I coefficienti per la routine di ritardo sono calcolati dalla lunghezza del ritardo ed i tempi di esecuzione delle varie istruzioni a velocità di clock standard per 8080 (cioè 2 MHz).

MVI - 3,5 μs
DCR - 2,5 μs
JNZ - 5,0 μs

Sono necessari due registri poichè uno non può fornire tutti i 9,1 millisecondi con la routine semplice.

La routine di ritardo può produrre facilmente qualunque multiplo di 1/8 del tempo di bit variando l'inizializzazione del Registro B.

Potrebbe essere usata anche la linea di uscita seriale (SOD) dell'8085. L'istruzione SIM carica l'uscita SOD di tipo latch con il bit 7 dell'Accumulatore se il bit 6 è ad 1. Gli altri bit dell'Accumulatore hanno funzioni di interrupt e devono avere appropriati valori (Vedere il Manuale dell'Utende dell'8085 oppure il capitolo successivo).

Il nuovo programma per l'8085 è:

ACCETTA DATI ED AZZERA IL BIT D'INIZIO

```
        MVI     D,11     ;CONTEGGIO = 11 BIT
        LDA     40H      ;ACCETTA DATI
        ANA     4        ;AZZERA IL BIT D'INIZIO
```

TRASMETTE UN BIT SERIALE ATTRAVERSO SOD

```
TBIT:   MOV     E,A      ;CONSERVA DATI
        MVI     A,IMASK  ;ACCETTA LA MASCHERA DI INTERRUPT
        RAR              ;DATI AL BIT DI USCITA SERIALE (BIT 7)
        SIM              ;DATI AL LATCH SERIALE
```

RITARDO DI 9,1 MS

```
        MVI     B,H      ;ATTRAVERSAMENTO 8 VOLTE
DLY:    MVI     C,243    ;RITARDO DI 1/8 DEL TEMPO DI BIT
DLY1:   DCR     C
        JNZ     DLY1
        DCR     B
        JNZ     DLY
```

SCORRIMENTO DATI E CONTEGGIO BIT

```
        STC              ;BIT DI ARRESTO = 1
        MOV     A,E      ;SCORRIMENTO DATI DI 1 BIT
        RAL
        DCR     D        ;CONTO ALLA ROVESCIA DI 11 BIT
        JNZ     TBIT
```

IMASK ha un '1' nel bit 7 per abilitare il caricamento del latch seriale dell'8085 (dopo RAL il bit 6 conterrà tale 1). I coefficienti per la routine di ritardo sono stati calcolati dal tempo di esecuzione delle varie istruzioni alla velocità di clock standard per l'8085 di 3 MHz. La routine di ritardo potrebbe preservare i Registri B e C con una istruzione PUSH B all'inizio e POP B alla fine. Non è necessaria una precisione elevata poichè la telescrivente non mantiene questa precisione elevata ed ogni trasmissione consiste di soli 11 bit.

ANA A azzera il Carry in modo che il bit d'inizio sia uno '0'; STC pone ad 1 il Carry cosicchè i bit di Arresto sono uni. Nessun'altra istruzione del programma influenza il Carry.

PROBLEMI

1) Separazione di Chiusure da una Tastiera non Codificata

Scopo: Il programma dovrebbe leggere ingressi da una tastiera 3 x 3 non codificata e posizionarli in un array. Il numero di ingressi richiesti è nella locazione di memoria 60 e l'array inizia nella locazione di memoria 61.

Separa una chiusura dalla successiva aspettando che la corrente di chiusura vada a zero. Si ricordi l'eliminazione di rimbalzo della tastiera (questo caso comporterà semplicemente un'attesa di pochi millisecondi).

Problema Campione:

(60) = 04
Gli Ingressi sono 7, 2, 2, 4
Risultato = (61) = 07
(62) = 02
(63) = 02
(64) = 04

2) Lettura di una Frase da una Tastiera Codificata

Scopo: Il programma dovrebbe leggere ingressi da una tastiera ASCII (7 bit con un bit Parità uguale a '0') e posizionarli in un array fino a che riceve un periodo ASCII (esadecimale 2E). L'array inizia nella locazione di memoria 60. Ogni ingresso è segnato da uno strobe come nell'esempio fornito a proposito di Una Tastiera Codificata.

Problema Campione:

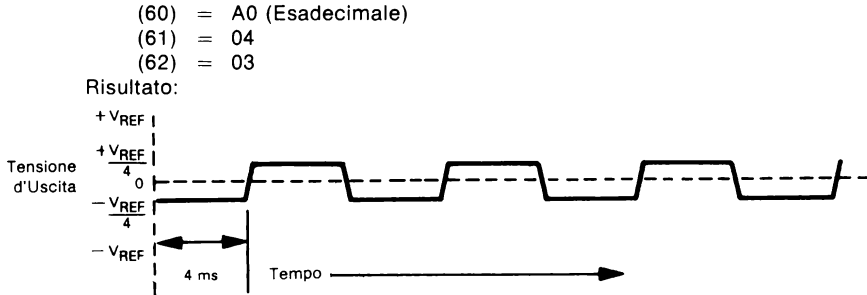
Gli ingressi sono H, E, L, L, 0.
Risultato = (60) = 48 H
(61) = 45 E
(62) = 4C L
(63) = 4C L
(64) = 4F 0
(65) = 2E

3) Un Generatore d'Onda Quadra Variabile in Ampiezza

Scopo: Il programma dovrebbe generare un'onda quadra come mostrato nella figura successiva da un Convertitore D/A. La locazione di memoria 60 contiene le ampiezze dell'onda, la locazione 61 la lunghezza di mezzo ciclo in millisecondi, e la locazione 62 il numero di cicli.

Si assume che un'uscita digitale di 80 esadecimale al convertitore si risolva in una uscita analogica di 0 volt. Per questo convertitore un'uscita digitale di X si risolve in una uscita analogica di VOUT = X-80/80x(-V_{REF}) volt.

Problema Campione:

(60) = A0 (Esadecimale)
(61) = 04
(62) = 03
Risultato:

Il potenziale di base è 80 esadecimale assunto 0 volt. Il fondo scala è 00 esadecimale per -VREF volt. Così A0 esadecimale:

$$V_{OUT} = \frac{A0 - 80}{80} \times -V_{REF} = \frac{-V_{REF}}{4}$$

Il programma produce tre impulsi di ampiezza VREF/4 con una mezza lunghezza di ciclo di 4 ms.

4) Operazione di Media di Letture Analogiche

Scopo: Il programma dovrebbe accettare quattro letture, da un convertitore A/D, separate di 10 millisecondi e posiziona la media nella locazione di memoria 60. Si assume che il convertitore A/D impiega meno di 100 microsecondi per eseguire la conversione in modo che il tempo di conversione può essere ignorato.

Problema Campione:

Le letture sono (esadecimali) 86, 89, 81, 84
Risultato = (60) = 85

5) Un Terminale a 30 Caratteri al Secondo

Scopo: Modificare le routine di trasmissione e di ricezione degli esempi forniti precedentemente per un terminale a 30 caratteri al secondo che trasferisce i dati ASCII con un bit di arresto e parità pari. Come si potrebbero scrivere le routine per maneggiare la dipendenza del terminale da un flag bit della locazione di memoria 40, cioè (40) = 0 per il terminale a 30 caratteri al secondo e (40) = 1 per il terminale a 10 caratteri al secondo?

Capitolo 12
INTERRUPT

Gli interrupt sono ingressi seriali che la CPU esamina come parte di ogni ciclo di istruzione. Questi ingressi permettono alla CPU di reagire ad eventi a livello hardware piuttosto che a livello software. Gli interrupt generalmente richiedono più hardware dell'I/O ordinario (programmati), ma forniscono una risposta più veloce e più immediata. È possibile reperire una discussione sugli interrupt nel Volume I di An Introduction To Microcomputers.

GENERALITÀ SUGLI INTERRUPT

Perchè vengono usati gli interrupt? Gli interrupt consentono eventi come allarmi, assenza di potenza, il passaggio di certe quantità di tempo ed a periferiche, che dispongono dati o sono pronti ad accettare dati, di acquisire l'attuazione immediata della CPU. Il programmatore non necessita di avere la CPU realmente controllata per questi eventi e nemmeno interessare la loro omissione. Un sistema di interrupt è come la suoneria di un telefono che suona quando è in corso una chiamata, cosicchè si raggiunge il telefono in risposta alla suoneria piuttosto che controllare la linea per vedere se qualcuno sta chiamando. Allo stesso modo la CPU non deve attendere che accada un certo evento ma piuttosto risponde all'evento quando si verifica. Naturalmente questa semplice descrizione si complica (proprio come un sistema telefonico) quando ci sono molti interrupt e compiti che non possono essere interrotti.

La realizzazione dei sistemi di interrupt è molto variabile. Tra le questioni che caratterizzano un particolare sistema ci sono:

CARATTERISTICHE DEI SISTEMI DI INTERRUPT

1) Quanti ingressi interrupt ci sono?
2) Quante sorgenti di interrupt ci sono?
3) Come riesce la CPU a determinare la sorgente di un interrupt se il numero di sorgenti supera il numero di ingressi?
4) Come risponde la CPU ad un interrupt?
5) La CPU può differenziare tra interrupt importanti e non importanti?
6) Come e quando il sistema di interrupt è abilitato?

Esistono molte risposte diverse a queste domande. Lo scopo di tutte le realizzazioni, comunque è di fare in modo che la CPU risponda rapidamente all'interrupt e riprenda la normale attività dopo la risposta.

Il numero di ingressi di interrupt sul chip della CPU determina il numero di risposte diverse che la CPU può fornire senza nessun hardware addizionale o software. Ogni ingresso può fornire una diversa risposta interna. Sfortunatamente la maggior parte dei microprocessori ha un numero molto piccolo (tipicamente uno e due) di ingressi interrupt separati. La risposta fondamentale della CPU ad un interrupt deve essere il trasferimento del controllo alla routine di servizio interrupt corretta e conservare il valore attuale del Contatore di Programma. La CPU deve perciò eseguire un Salto-Alla-Subroutine od Istruzione di Chiamata con l'inizio della routine di servizio interrupt al suo indirizzo. In questo modo verrà conservato l'indirizzo di ritorno nello Stack e trasferito il controllo alla routine di servizio interrupt. La quantità di hardware esterno richiesto per originare la risposta è molto variabile. Alcune CPU generano internamente l'istruzione e l'indirizzo; altre richiedono hardware esterno per formarli. La CPU può generare soltanto un'istruzione o indirizzo diverso per ogni ingresso distinto.

POLLING

VETTORIZZAZIONE

Se il numero di dispositivi che generano interrupt è superiore al numero di ingressi, la CPU richiederà ulteriore hardware o software per identificare la sorgente dell'interrupt. Nel caso più semplice il software può consistere di una routine di polling (registrazione) che controlla lo stato del dispositivo che deve originare l'interrupt. L'unico vantaggio di un tale sistema su un normale polling è che la CPU sa che almeno un ingresso è attivo. La soluzione alternativa è quella introdurre hardware addizionale per fornire un unico ingresso dati (o vettore) di ogni sorgente. Le due alternative possono essere unite; i vettori possono identificare gruppi d'ingressi che la CPU può differenziare mediante polling.

PRIORITÀ

Un sistema di interrupt che può differenziare tra interrupt importanti è detto «sistema a priorità di interrupt». L'hardware interno può fornire tanti livelli di priorità quanti sono gli ingressi. L'hardware esterno può fornire livelli addizionali per mezzo dell'impiego di un registro di priorità e comparatore. Un sistema a priorità di interrupt può avere bisogno di una via speciale per maneggiare gli interrupt a bassa priorità che possono essere ignorati per lunghi periodi di tempo.

ABILITAZIONE E DISABILITAZIONE DI INTERRUPT

INTERRUPT NON MASCHERABILE

La maggior parte dei sistemi di interrupt può essere abilitata o disabilitata. Infatti la maggior parte delle CPU automaticamente disabilitano gli interrupt quando viene eseguito un RESET (cosicchè il programmatore può configurare il sistema di interrupt) ed accettando un interrupt (cosicchè un interrupt non interromperà la sua routine di servizio). Il programmatore può desiderare disabilitare gli interrupt nella preparazione od elaborazione di dati, nell'esecuzione di un ciclo di timing oppure eseguendo un'operazione a parola multipla. Un interrupt che non può essere disabilitato (talvolta detto «interrupt non mascherabile») può essere conveniente per l'interrupt di potenza insufficiente (caduta alimentazione).

SVANTAGGI DEGLI INTERRUPT

I vantaggi degli interrupt sono ovvi ma ci sono anche degli svantaggi. Questi comprendono:

1) I sistemi di interrupt possono richiedere una grande quantità di hardware extra.
2) Gli interrupt richiederanno il trasferimento dei dati sotto il controllo del programma attraverso la CPU.
3) Gli interrupt sono ingressi casuali che rendono difficoltoso il debugging ed il testing. Gli errori possono ricorrere sporadicamente e perciò possono essere difficili da trovare (per una discussione di progettazione con interrupt vedere Baldridge, R.L., «Interrupts Add Power, Complexity to µC-System Design», EDN, 5 Agosto 1977, pp 67-73).

SISTEMA DI INTERRUPT DELL'8080

La risposta interna dell'8080 ad un interrupt è molto semplice. Il sistema di interrupt consiste di:

1) Un singolo ingresso di richiesta interrupt attivo-alto.
2) Un flip-flop di abilitazione interrupt, il cui stato è disponibile come uscita esterna (INTE).
3) Un bit di stato, INTERRUPT ACKNOWLEDGE (INTA) (Riconoscimento Interrupt) che la CPU posiziona sulla linea 0 del Bus Dati all'inizio di ogni ciclo di macchina.

RISPOSTA ALL'INTERRUPT PER L'8080

L'8080 controlla la linea di interrupt alla fine di ogni ciclo di istruzione. Se la linea di richiesta interrupt è attiva e gli interrupt sono abilitati, la risposta è la seguente (Vedere Volume II di An Introduction To Microcomputers):

1) La CPU disabilita il sistema di interrupt.
2) La CPU esegue un ciclo di macchina per INTERRUPT ACKNOWLEDGE.
   Questo ciclo è contraddistinto dal bit di stato INTA = 1. DBIN è attivato durante il ciclo cossichè la CPU preleverà una istruzione, ma MEMR = 0, cosicchè la CPU non attiverà la memoria.

L'ardware esterno può fornire il resto della risposta.

L'ISTRUZIONE RESTART (RST)

L'ISTRUZIONE RESTART

Il set d'istruzioni dell'8080 comprende un'istruzione speciale, Restart (Riparti) (RST), intesa per l'impiego con interrupt. Restart è una istruzione di chiamata di una parola che conserva il valore attuale del Contatore di Programma nello Stack e salta all'indirizzo specificato nell'istruzione. La Tabella 12-1 contiene le varie istruzioni Restart ed i loro indirizzi di destinazione.

Tabella 12-1
L'Istruzione Restart

Forma Mnemonica	Forma Binaria	Forma Esadecimale	Indirizzo di Destinaz. (Esadec.)
RST 0	11000111	C7	0000
RST 1	11001111	CF	0008
RST 2	11010111	D7	0010
RST 3	11011111	DF	0018
RST 4	11100111	E7	0020
RST 5	11101111	EF	0028
RST 6	11110111	F7	0030
RST 7	11111111	FF	0038

Solo per l'8085 (ingressi separati)

RST 5.5			002C
RST 6.5			0034
RST 7.5			003C

RST è conveniente in sistemi di interrupt per le ragioni seguenti:

1) È una istruzione di una parola e richiede solo un ciclo di acquisizione.
2) Fornisce otto diversi indirizzi di destinazione o vettori.
3) I suoi vettori sono abbastanza separati per permettere alle istruzioni di salto di raggiungere le effettive routine di servizio.
4) È facile da formare poichè 5 bit sono sempre '1'. Un codificatore da 8 a 3 può fornire gli altri tre bit abbastanza facilmente.

RST ha i seguenti svantaggi:

1) Non può fornire più di 8 vettori.
2) I suoi vettori sono tanto separati da permettere spazio per intere routine di servizio interrupt.
3) I suoi vettori si trovano in un'area di memoria.
4) RST 0 ha lo stesso indirizzo di destinazione dell'ingresso RESET ed è perciò molto difficoltosa da impiegare. Il sistema necessita di hardware per distinguere tra RESET e RST 0 poichè non è possibile distinguere unicamente con software.

PRODUZIONE DI UNA SINGOLA ISTRUZIONE RESTART

Il progettista può produrre una singola istruzione RST (RST7) mediante uno qualsiasi dei seguenti metodi:

1) Impiegare dei resistori di pullup cosicchè le linee Bus Dati sono tutte al livello alto quando nessuna sorgente sta pilotando il Bus (Vedere Fig. 12-1).
2) Collegare l'uscita $\overline{INTA}$ di un Sistema di Controllo 8228 a +12V attraverso un resistore da 1K Ω (Vedere Figura 12-2)
3) Usare una porta I/0 8212 come porta di istruzione interrupt (Vedere Figura 12-3) con i suoi ingressi collegati al livello alto e la sua linea di selezione collegata ad $\overline{INTA}$ (formata chiudendo INTA e DBIN).

Se c'è più di un dispositivo di interrupt, la routine di servizio interrupt, iniziante alla locazione di memoria 38 esadecimale deve identificare la sorgente considerata come un I/O ordinario.

Un codificatore da 8 a 3 può fornire tutte le otto istruzioni RST come mostrato in Figura 12-4. Si ricordi che gli ingressi e le uscite di un codificatore 74LS148 sono attive al livello basso. Come risultato, un livello basso sull'ingresso $\overline{R}_0$ produce l'istruzione RST 7 (Vedere Tabella 12-2) e l'ingresso $\overline{R}_7$ produce l'istruzione RST 0 che ha lo stesso indirizzo di RESET. Il codificatore differenzia solo tra ingressi attivi contemporanei e produce un'uscita che corrisponde all'ingresso a priorità più alta.

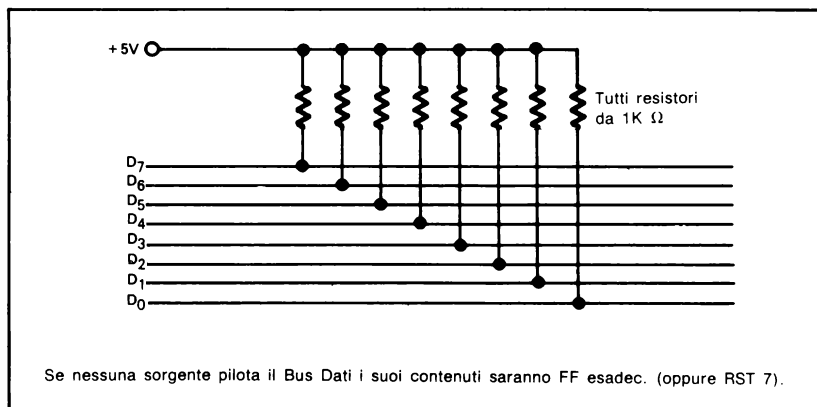



Figura 12-1
Impiego Dei Resistori Di Pullup Per Formare L'Istruzione RST7

Con questo circuito il Controllore di Sistema 8228 forzerà il Bus Dati ad FF per un tempo opportuno in risposta ad INTA = 1.

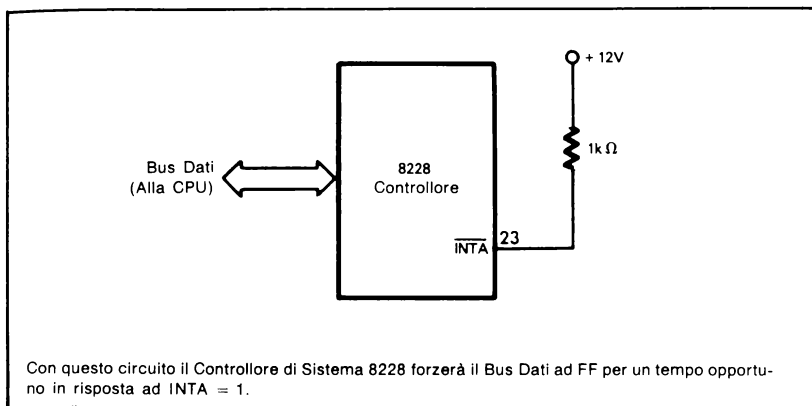



Figura 12-2
Impiego Del Controllore Di Sistema 8228 Per Formare L'Istruzione RST7

In risposta ad $\overline{INTA}$ attivo, la porta 8212 posiziona FF esadec. (RST 7) sul Bus Dati.

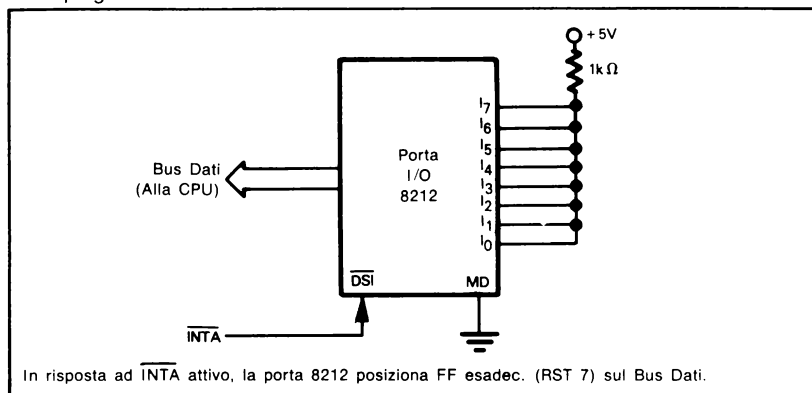



Figura 12-3
Impiego Della Porta I/O 8212 Per Formare L'Istruzione RST7

Il codificatore di priorità fornisce tre bit di uscita attivi al livello basso ad una porta 8212. Il risultato è il posizionamento di una delle otto istruzioni RST del Bus Dati alla CPU in risposta al segnale $\overline{INTA}$.

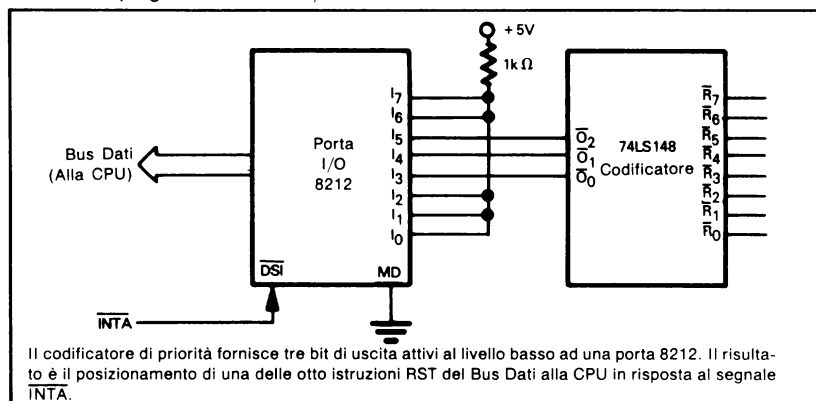



Figura 12-4
Formazione Di Otto Istruzioni RST Con Un Codificatore Di Priorità

SISTEMA DI INTERRUPT DELL'8085

Il sistema di interrupt dell'8085 differisce da quello dell'8080 perchè ha quattro ingressi interrupt addizionali. I quattro nuovi ingressi hanno tutti priorità sull'ingresso regolare di interrupt (INTR); essi sono:

SISTEMA INTERRUPT 8085

1) RST 5.5 che forza una chiamata all'indirizzo 2C (esadec.).
2) RST 6.5 che forza una chiamata all'indirizzo 34 (esadec.).
3) RST 7.5 che forza una chiamata all'indirizzo 3C (esadec.).
4) TRAP (ovvero interrupt non mascherabile) che forza una chiamata all'indirizzo 24 (esadec.). Questo ingresso può essere impiegato come interrupt per potenza insufficiente (caduta alimentazione).

L'ordine di priorità (dal più alto al più basso) è TRAP, RST7.5, RST6.5, RST5.5 ed INTR.

RST5.5, RST6.5, RST7.5 possono essere mascherati (disabilitati) con una istruzione SIM. L'istruzione SIM impiega i bit dell'Accumulatore come segue:

ISTRUZIONE SIM

- Bit 7 = Dati Uscita Seriale (SOD)
- Bit 6 = 1 per abilitare i dati dell'uscita seriale, cioè caricare il latch d'uscita.
- Bit 5 non è impiegato
- Bit 4 = 1 per ripristinare la richiesta RST 7.5
- Bit 3 = 1 per caricare la maschera di interrupt dal bit 0 al bit 2
- Bit 0-2 = 0 per abilitare gli interrupt da RST5.5 a RST 7.5 rispettivamente

Similmente l'istruzione RIM può determinare i valori delle maschere e rilevare se non c'è nessun interrupt in corso. RIM carica l'Accumulatore come segue:

ISTRUZIONE RIM

- Bit 7 = Dati Ingresso Seriale (SID)
- Bit 4-6 = 1 per richieste in corso ai livelli 5.5, 6.5 e 7.5 rispettivamente
- Bit 3 = 1 per abilitare l'intero sistema di interrupt (eccetto TRAP)
- Bit 0-2 sono i valori attuali delle maschere di interrupt (0 significa abilitato) per RST 5.5, RST 6.5, ed RST 7.5 rispettivamente

UNITÀ DI CONTROLLO DELLA PRIORITÀ DI INTERRUPT 8214

Un'unità di Controllo della Priorità di Interrupt 8214 può fornire un sistema d'interrupt vettorizzato ad 8 livelli pressochè completo. Questo dispositivo (vedere Figura 12-5) contiene:

UNITÀ DI CONTROLLO INTERRUPT 8214

1) Un codificatore di priorità da 8 a 3.
2) Un registro di stato corrente.
3) Un comparatore di priorità.
4) Vari ingressi ed uscite per il controllo e l'espansione.

Alcune caratteristiche del dispositivo 8214 sono (Vedere anche il Volume II di An Introduction To Microcomputers):

1) Le uscite sono attive al livello basso; le istruzioni RST prodotte sono fornite dalla Tabella 12-2.
2) Il registro di Stato corrente è anch'esso attivo al livello basso cosicchè 111 è la priorità più bassa e 000 la più alta. La Tabella 12-3 mostra quali priorità sono consentite dai diversi valori del Registro di Stato.
3) $\overline{SGS}$ (Selezione Gruppo di Stato) è il bit 3 del registro di Stato. Se questo bit è al livello basso viene attivato il confronto di stato e saranno accettati solo gli interrupt precedenti il livello dato dal registro di Stato. Se $\overline{SGS}$ è al livello alto il confronto di stato è aggirato in modo che saranno accettati gli interrupt al livello 0.
4) La CPU può posizionare un nuovo valore nel registro di Stato indirizzandolo come una porta di uscita attraverso $\overline{ECS}$ (Abilita Lo Stato Corrente, attivo al livello basso).
5) Poichè il registro di Stato è solo una porta d'uscita la CPU non può determinare direttamente i suoi contenuti.

Si noti che i contenuti del registro di Stato dell'8214 devono essere controllati. Occorre posizionare in esso un valore iniziale (tipicamente $\overline{SGS}$ = 1, per esempio OF_{16}) prima dell'abilitazione di interrupt. Come parte di ogni routine di servizio interrupt occorre posizionare il valore di priorità corretto nel registro di Stato prima della riabilitazione e deve immagazzinare il vecchio valore di priorità nel registro di Stato prima del ritorno al programma interrotto.

Ingresso Attivo Richiesto più Alto	Istruzione RST	Indirizzo di Destinaz. (Esadec.)
0	RST 7	0038
1	RST 6	0030
2	RST 5	0028
3	RST 4	0020
4	RST 3	0018
5	RST 2	0010
6	RST 1	0008
7	RST 0	0000
Si ricordi che le uscite del codificatore sono attive al livello basso.		

Tabella 12-2
Istruzioni RST Prodotte Da Diversi Ingressi Di Richiesta

Valori del Registro di Stato		Interrupt Consentiti
$\overline{SGS}$	B	
0	0	Nessuno
0	1	Richiesta 7
0	2	Richiesta 6 o sopra
0	3	Richiesta 5 o sopra
0	4	Richiesta 4 o sopra
0	5	Richista 3 o sopra
0	6	Richiesta 2 o sopra
0	7	Richiesta 1 o sopra
1	Nessuno	Tutti

Tabella 12-3
Interrupts Consentiti Per Diversi Valori Del Registro Di Stato

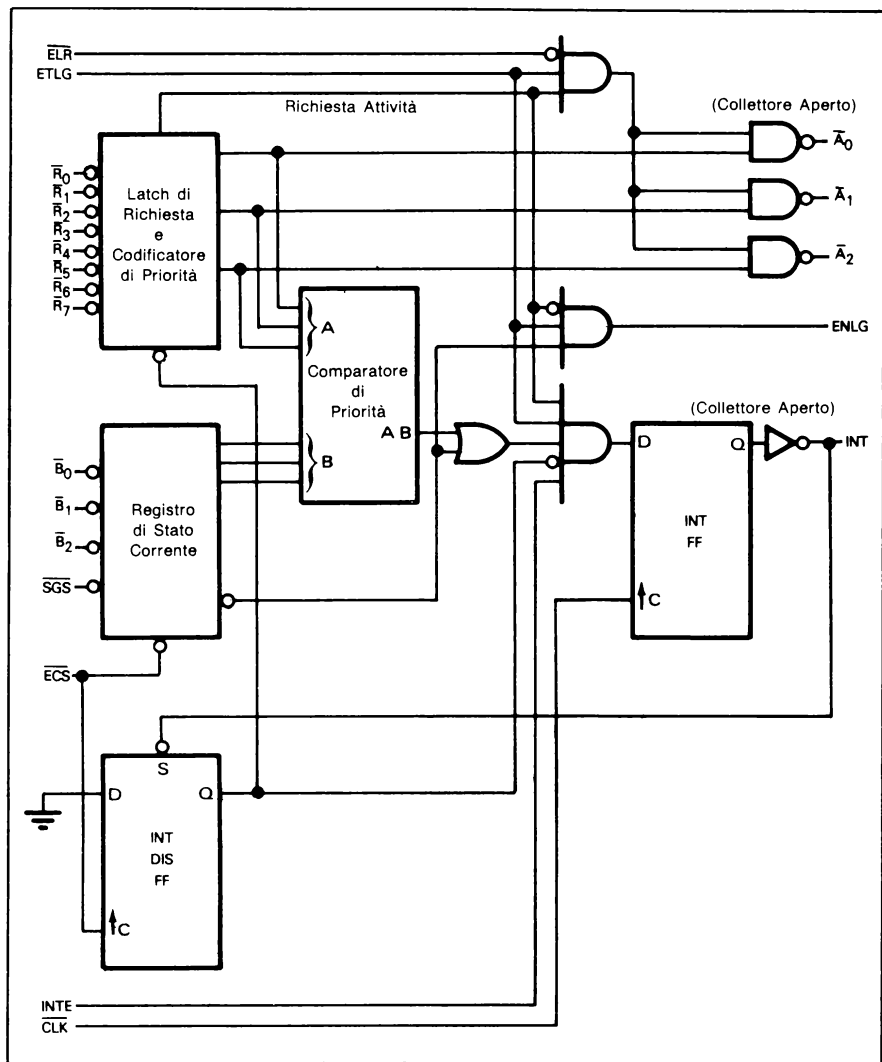



Figura 12-5
L'Unità Di Controllo della Priorità Di Interrupt

$\overline{SGS}$ deve essere '1' per abilitare interrupt all'ingresso $\overline{R_0}$, poichè qualsiasi livello di priorità permette solo gli interrupt sopra quel livello. Questa procedura è necessaria per evitare che un ingresso di richiesta interrompa la propria routine di servizio.

La Figura 12-6 mostra le connessioni richieste per formare un controllore di interrupt ad otto-livelli da una unità di controllo 8214 ed una porta I/O 8212.

I vantaggi del dispositivo 8214 sono:

1) Fornisce otto livelli di interrupt a priorità vettorizzata con un hardware minimo.
2) Non richiede nessun timing od altra circuiteria.
3) Contiene il registro di priorità ed il comparatore.

Gli svantaggi del dispositivo 8214 sono:

1) Lo schema della priorità è fisso.
2) Gli interrupt a bassa priorità possono essere ignorati.
3) L'espansione del sistema interrupt oltre gli otto livelli è difficoltosa.
4) I vettori di interrupt si trovano in un'area fissa di memoria.
5) La CPU non può leggere i contenuti del registro di priorità. Perciò, il programma deve conservare una copia della priorità corrente su RAM.

Nonostante questi svantaggi il dispositivo 8214 è un controllore completo, semplice, a basso costo per sistemi di interrupt che non hanno un numero di ingressi elevati.

REGOLATORE DI INTERRUPT PROGRAMMABILE 8259

REGOLATORE DI INTERRUPT 8259

Il Regolatore di Interrupt Programmabile 8259 (Vedere Figura 12-7) supera alcuni degli svantaggi del dispositivo 8214. Questo regolatore ha le seguenti caratteristiche:

1) Consente diversi metodi di priorità, compresa la priorità a rotazione (cioè il livello di priorità ruota dopo ogni servizio), per assicurare che tutti gli interrupt siano eventualmente asserviti.
   Il metodo di priorità può essere cambiato sotto il controllo del programma.
2) Può posizionare i vettori di interrupt dovunque nella memoria (impiegando un'area di 32 o 64 byte)
3) Un numero di questi dispositivi maggiore di otto può essere combinato per fornire 64 livelli di interrupt vettorizzati.

Il maggior svantaggio del regolatore 8259 è la richiesta di programmazione e la sua dipendenza dal Regolatore di Sistema 8228 per fornire gli impulsi di timing che porta ad un'istruzione di Chiamata completa sul Bus Dati. Il Volume II di An Introduction To Microcomputers discute il dispositivo 8259.

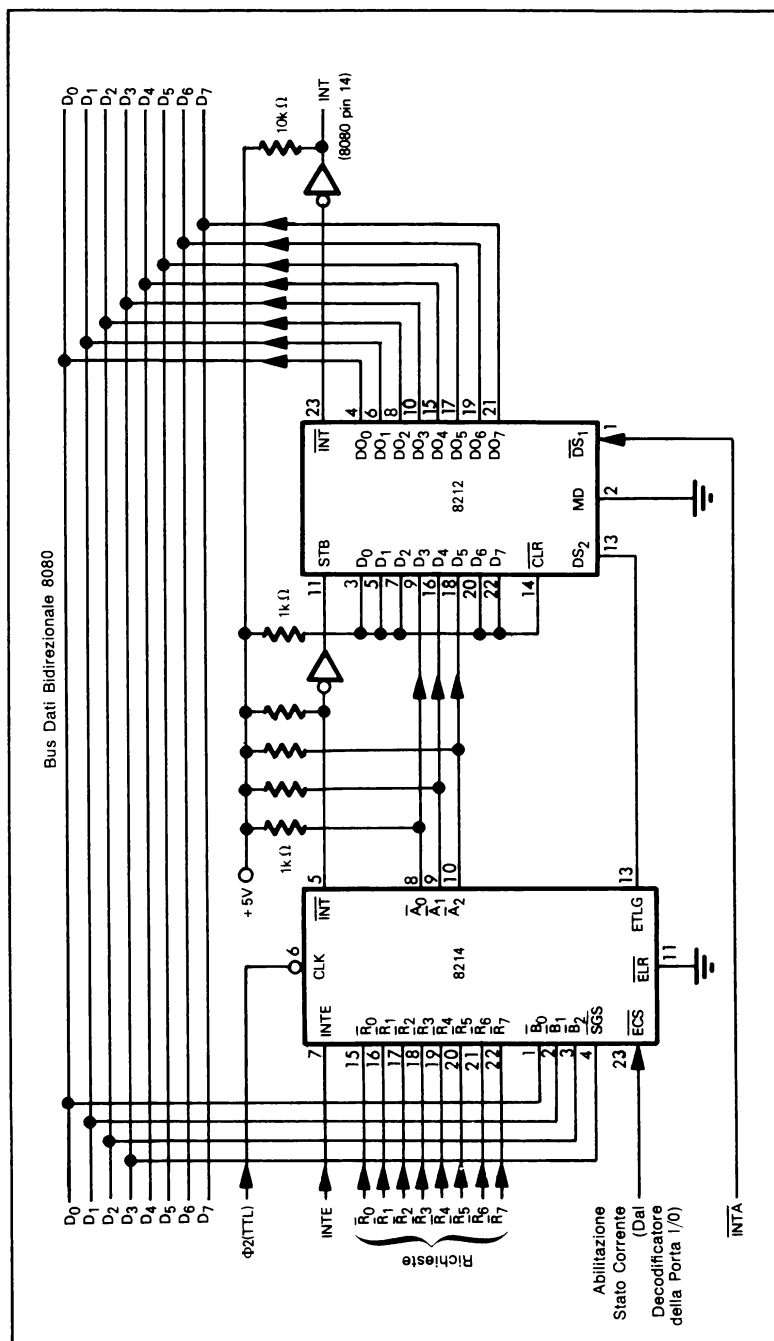



Figura 12-6
L'Unità Di Controllo Della Priorità Di Interrupt 8214
Impiegata Come Controllore Ad 8 Livelli

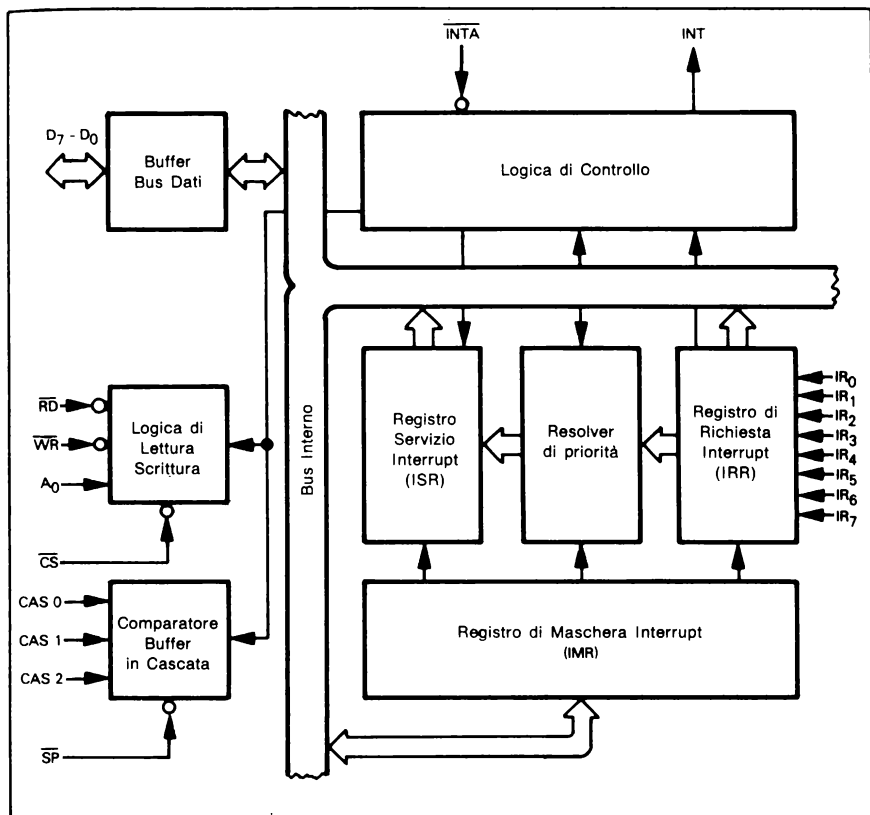



Figura 12-7
Il Regolatore Di Interrupt Programmabile 8259

ESEMPI

Uno Startup Interrupt

Scopo: Avere la CPU in stato di attesa fino a che è presente lo startup interrupt.

STARTUP INTERRUPT

Molti sistemi rimangono inattivi fino a che l'operatore li fa partire ovvero ricevono un segnale DATA READY. Quando è presente il RESET della fornitura di potenza in tali sistemi, il software deve inizializzare il Puntatore dello Stack, abilitare gli interrupt ed eseguire un'istruzione HLT. (Si ricordi che un RESET disabilita gli interrupt). Nel diagramma di flusso la decisione per vedere se startup è attivo è fatto in hardware (cioè attraverso l'esame da parte della CPU dell'ingresso interrupt fatto internamente) piuttosto che in software.

La Figura 12-8 mostra la circuiteria richiesta dall'interrupt. Si noti che l'ingresso startup sarà attivo al livello alto poichè il flip-flop di richiesta servizio dell'8212 risponde ad una transizione da basso ad alto sull'ingresso STB. Anche l'uscita $\overline{INT}$ dall'8212 deve essere prima invertita essendo connessa all'ingresso interrupt attivo al livello alto dell'8080.

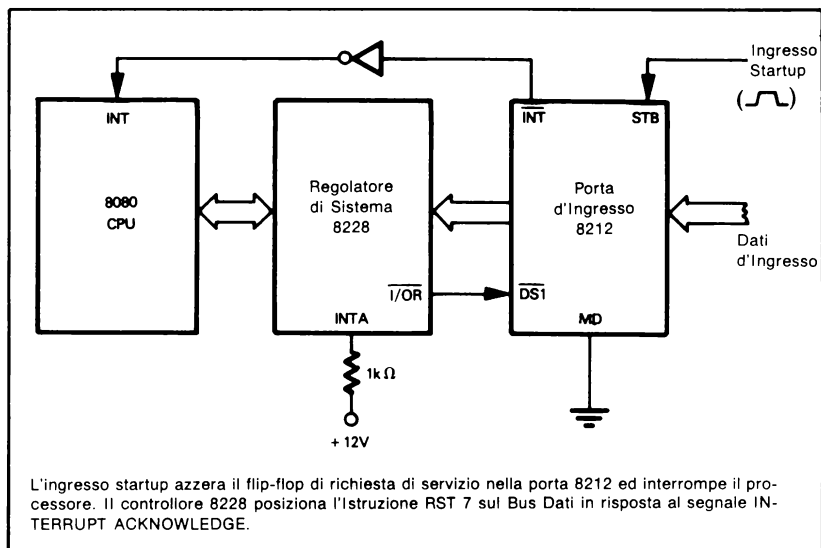
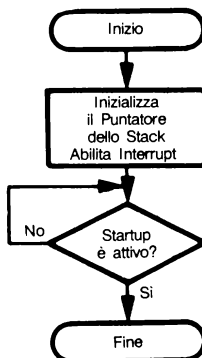



Figura 12-8
Un Interrupt Ad Ingresso Singolo

Diagramma di Flusso:

Si noti che l'hardware, piuttosto che il software, determina se lo startup è attivo.

Programma Sorgente:

```
RESET   EQU    0
        ORG    RESET
        LXI    SP,100H  ;PONI LO STACK ALLA FINE DELLA MEMORIA
        EI              ;ABILITA INTERRUPT
        HLT             ;ED ATTENDI
        ORG    INTRP
        IN     PORT     ;AZZERA INTERRUPT
        LXI    SP,100H  ;RI-INIZIALIZZA IL PUNTATORE DELLO STACK
HERE    JMP    HERE
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00		LXI	SP,100H	31
01				00
02				01
03		EI		FB
04		HLT		76
INTRP		IN	PORT	DB
+1				PORT
+2		LXI	SP,100H	31
+3				00
+4				01
+5	HERE:	JMP	HERE	C3
+6				INTRP+5
+7				

Il programma deve inizializzare il Puntatore dello Stack poichè l'istruzione RST impiega lo Stack. Ogni iterazione ri-inizializza il Puntatore dello Stack cosicchè lo Stack non aumenta.

Il programma deve inizializzare il Puntatore dello Stack ed il puntatore buffer prima dell'abilitazione di interrupt in modo che la routine di servizio ha lo spazio per i dati e l'indirizzo di ritorno.

LOCAZIONE DELLA ROUTINE DI SERVIZIO INTERRUPT

La esatta locazione della routine di servizio interrupt varia al variare del microcomputer. Se si ha a disposizione un microcomputer senza monitor si può iniziare la routine di servizio interrupt dove l'hardware esterno dirige la CPU. Indirizzi convenienti da usare sono 38 esadecimale nel sistema 8080 (l'interrupt a priorità più bassa nei sistemi basati sull'8214 ed il solo vettore nei sistemi senza regolatori di interrupt) oppure 3C esadecimale nei sistemi 8085 (il vettore per l'ingresso RST 7.5). Questi indirizzi sono convenienti poichè non ci sono vettori nella memoria sopra di essi.

MANIPOLAZIONE DI INTERRUPT MEDIANTE MONITOR

Se si ha a disposizione un microcomputer con monitor, quest'ultimo spesso occuperà il RESET e gli indirizzi di servizio interrupt. Esso poi fornirà gli indirizzi di servizio nei quali si deve posizionare sia le routine di servizio ovvero gli indirizzi di quelle. Una tipica routine di monitor potrebbe essere:

```
MONIN:  LHLD    USINT   ;ACCETTA L'INDIRIZZO UTENTE
                        ;PER LA ROUTINE DI SERVIZIO INTERRUPT
        PCHL            ;E SALTA AD ESSA
```

Occorre posizionare l'indirizzo della routine di servizio delle locazioni di memoria USINT ed USINT+1. Si ricordi che MONIN è un indirizzo del programma monitor.

È possibile includere il caricamento delle locazioni di memoria USINT ed USINT+1 nel programma principale, cioè:

```
        LXI     H,INTRP ;ACCETTA L'INDIRIZZO DI PARTENZA DELLA
                        ;ROUTINE DI SERVIZIO
        SHLD    USINT   ;IMMAGAZZINA LO STESSO COME INDIRIZZO UTENTE
```

Queste istruzioni vengono prima dell'abilitazione degli interrupt.

In questo esempio, l'indirizzo di ritorno che l'8080 immagazzina nello Stack non è conveniente. Comunque il programma principale dovrà inizializzare il Puntatore dello Stack in modo che esiste una posizione definita dove allocare tale indirizzo. L'istruzione LXI SP non è necessaria se il monitor nel microcomputer a disposizione dirige il Puntatore dello Stack.

La routine di servizio scarica l'indirizzo di ritorno mediante una semplice ri-inizializzazione del Puntatore dello Stack. Un'alternativa potrebbe essere quella di incrementare due volte il Puntatore dello Stack:

```
INX     SP
INX     SP
```

Si ricordi che accettando automaticamente un interrupt si disabilita il sistema di interrupt. Questo permette alla routine di startup reale di configurare il sistema di interrupt prima dell'abilitazione di interrupt. Si noti che si può disattivare lo startup interrupt nella routine di servizio (con o senza l'istruzione IN PORT) oppure esso sarà interrotto quando il sistema di interrupt è riabilitato.

La realizzazione delle istruzioni EI (Abilita Interrupt) e DI (Disabilita Interrupt) differisce nell'8080. DI ha effetto immediatamente dopo la sua esecuzione EI ha effetto dopo l'esecuzione dell'istruzione successiva. Il ragionamento su questo fatto è discusso al Capitolo 3 sotto la descrizione EI.

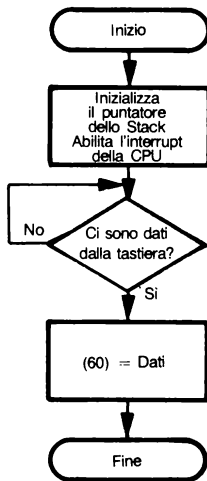
Un Interrupt da Tastiera

INTERRUPT DA TASTIERA

Scopo: Il computer attende un interrupt da tastiera e posiziona i dati dalla tastiera nella locazione di memoria 60.

Problema Campione:

Dati di Tastiera = 06
Risultato = (60) = 06

Diagramma di Flusso:

Programma Sorgente:

```
RESET   EQU     0
        ORG     RESET
        LXI     SP,100H  ;PONI IL PUNTATORE DELLO STACK ALLA FINE DELLA
                         ;MEMORIA
        EI               ;ABILITA INTERRUPT
HERE:   JMP     HERE     ;ED ATTENDI
        ORG     INTRP
        IN      PORT     ;ACCETTA DATI DALLA TASTIERA
        STA     60H      ;SALVA I DATI PROVENIENTI DALLA TASTIERA
        EI               ;RIABILITA INTERRUPT
        RET
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)	Contenuti di Mem. (Esadec.)
00	RESET: LXI SP,100H	31
01		00
02		01
03	EI	FB
04	HERE: JMP HERE	C3
05		04
06		00
INTRP	IN PORT	DB
+1		PORT
+2	STA 60H	32
+3		60
+4		00
+5	EI	FB
+6	RET	C9

L'istruzione JMP HERE è un salto all'istruzione stessa impiegata per rappresentare il programma principale. Dopo che gli interrupt sono abilitati in un sistema di lavoro, il programma principale viene eseguito fino a che un interrupt non interviene e poi riprende l'esecuzione dopo che è stata completata la routine di servizio interrupt.

VARIAZIONE DELL'INDIRIZZO DI RITORNO

L'istruzione RET alla fine della routine di servizio ri-trasferisce il controllo all'istruzione JMP. Se si vuole evitare questo si può aumentare semplicemente il Contatore di Programma nello Stack, cioè:

```
POP     H       ;ACCETTA L'INDIRIZZO DI RITORNO
INX     H       ;INCREMENTA L'INDIRIZZO DI RITORNO
INX     H
INX     H
PUSH    H       ;RIPOSIZIONALO NELLO STACK
```

L'istruzione RET ora trasferirà il controllo all'istruzione successiva alla JMP.

Poichè l'8080 non conserva automaticamente i suoi registri essi possono essere impiegati per passare i parametri ed i risultati tra il programma principale e la routine di servizio interrupt. Così si potrebbero inviare i dati nell'Accumulatore anzichè nella locazione di memoria 60. Questa comunque è una pratica molto pericolosa che andrebbe evitata in tutti ma soprattutto nei sistemi più semplici. In generale nessuna routine di servizio non altererebbe mai qualunque registro se i contenuti di quel registro sono stati conservati precedentemente la sua esecuzione e quindi ri-immagazzinati dopo il completamento della routine.

Si noti che si deve esplicitamente riabilitare gli interrupt alla fine della routine di servizio poichè il processore disabilita il sistema di interrupt quando accetta un interrupt. La lettura della porta disabilita l'interrupt.

RIEMPIMENTO DI UN BUFFER ATTRAVERSO INTERRUPT

Un'alternativa per una tastiera potrebbe essere di mantenere l'interruzione fino a che riceve una frase intera, per esempio terminante con un ritorno carrello ('CR'). Il programma principale e la routine di servizio potrebbero essere:

```
RESET:  LXI     SP,100H    ;STACK ALLA FINE DELLA MEMORIA
        LXI     H,60H      ;INIZIA IL PUNTATORE DEL BUFFER
        EI                 ;ABILITA INTERRUPT
HERE:   JMP     HERE       ;ED ATTENDE
;
INTRP:  IN      PORT       ;ACCETTA DATI DALLA TASTIERA
        MOV     M,A        ;CONSERVA NEL BUFFER
        CPI     CR         ;IL DATO È UN RITORNO CARRELLO?
        JZ      ENDB       ;SÌ, FINE DELLA FRASE
        INX     H          ;NO, INCREMENTA IL PUNTATORE
                           ;DEL BUFFER
        EI                 ;RIABILITA GLI INTERRUPT
        RET
ENDB:   POP     D          ;DISPONE L'INDIRIZZO DI RITORNO
        JMP     KBDCODE    ;CONTINUA CON GLI INTERRUPT
                           ;DISABILITATI
```

BUFFERING DOPPIO

Quando il processore riceve un ritorno carrello, esso disabilita il sistema di interrupt mentre manipola la frase. Un approccio alternativo potrebbe essere quello di riempire un altro buffer mentre si manipola il primo; quello approccio è chiamato buffering doppio.

Nelle applicazioni pratiche la CPU dovrebbe eseguire altri compiti tra gli interrupt. Essa potrebbe per esempio eseguire un edit, muovere o trasmettere una linea da un buffer mentre l'interrupt stava riempendo un altro buffer.

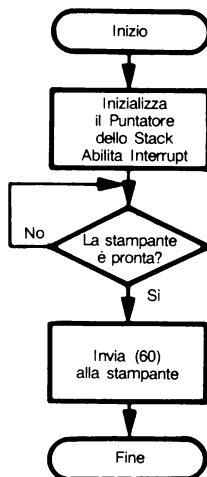
Un Interrupt da Stampante

INTERRUPT DA STAMPANTE

Scopo: Il computer attende un interrupt da stampante ed invia a quest'ultima i dati della locazione di memoria 60.

Problema Campione:

(60) = 41
Risultato = La stampante riceve 41 (ASCII A) quando è pronta.

Diagramma di Flusso:

Programma Sorgente:

```
RESET   EQU   0
        ORG   RESET
        LXI   SP,100H  ;PONE LO STACK ALLA FINE DELLA MEMORIA
        EI             ;ABILITA INTERRUPT
HERE:   JMP   HERE     ;ED ATTENDE
        ORG   INTRP
        LDA   60H      ;ACCETTA DATI
        OUT   PORT     ;INVIA ALLA STAMPANTE
        EI             ;RIABILITA INTERRUPT
        RET            ;ED ATTENDE
```

Programma Oggetto

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00	RESET:			
		LXI	SP,100H	31
01				00
02				01
03		EI		FB
04	HERE:	JMP	HERE	C3
05				04
06				00
INTRP		LDA	60H	3A
+1				60
+2				00
+3		OUT	PORT	D3
+4				PORT
+5		EI		FB
+6		RET		C9

Anche la scrittura dei dati nella porta di interruzione origina la disattivazione dell'interrupt.

RIEMPIMENTO DI UN BUFFER CON INTERRUPT

Anche qui si potrebbe avere che la stampante continua l'interrupt fino a che non è stata trasferita un'intera fase. Il programma principale e la routine di servizio interrupt potrebbero essere:

```
RESET:  LXI   SP,100H  ;STACK ALLA FINE DELLA MEMORIA
        LXI   H,60H    ;INIZIO PUNTATORE BUFFER
        EI             ;ABILITA INTERRUPT
HERE:   JMP   HERE     ;ED ATTENDE
;
INTP:   MOV   A,M      ;ACCETTA CARATTERE
                       ;DAL BUFFER
        OUT   PORT     ;INVIA CARATTERE ALLA STAMPANTE
        CPI   CR       ;IL CARATTERE È UN
                       ;RITORNO CARRELLO?
        JZ    ENDB     ;SÌ, FINE DELLA FRASE
        INX   H        ;NO, INCREMENTA IL PUNTATORE
                       ;DEL BUFFER
        EI             ;RIABILITA INTERRUPT
        RET
ENDB:   POP   D        ;DISPONE DELL'INDIRIZZO DI RITORNO
        JMP   MAIN     ;CONTINUA CON INTERRUPT DISABILITATI
```

La routine di servizio disabilita gli interrupt dopo aver completato il trasferimento della frase. Come nel caso precedente la CPU potrebbe eseguire altri compiti tra gli interrupt.

Un Interrupt da Clock in Tempo Reale

Scopo: Il computer attende per un interrupt da clock in tempo reale.

CLOCK IN TEMPO REALE

Un clock in tempo reale fornisce semplicemente una serie regolare di impulsi che possono essere impiegati come riferimento del tempo. Gli interrupt da un clock in tempo reale possono essere conteggiati per fornire qualunque intervallo di tempo richiesto.

Un clock in tempo reale può essere ottenuto suddividendo il clock della CPU impiegando un generatore di clock separato, ovvero un timer programmabile come il dispositivo 8253 per i microcomputer basati sull'8080, oppure mediante l'impiego di una sorgente esterna come la frequenza di rete in CA.

FREQUENZA DEL CLOCK IN TEMPO REALE

Si noti il compromesso che sorge nella determinazione della frequenza clock in tempo reale. Una frequenza elevata (diciamo 10KHz) permette di produrre un grande range di intervalli di tempo ad alta precisione. D'altra parte il tempo impiegato dal processore nel conteggio degli interrupt di clock può essere considerevole ed il conteggio supererà velocemente la capacità di un singolo registro ad 8 bit o locazione di memoria. La scelta della frequenza dipende dalla precisione e dalle richieste di timing dell'applicazione che si considera.

SINCRONIZZAZIONE COL CLOCK IN TEMPO REALE

La sincronizzazione del processore con il clock in tempo reale costituisce un problema. Iniziando da un punto casuale del periodo di clock si introdurrà un errore negli intervalli di timing. Alcuni metodi di sincronizzazione sono:

1) Inizio contemporaneo della CPU e del clock. Un RESET ovvero uno startup interrupt possono far partire il clock e la CPU.
2) Consentire alla CPU di far partire ed arrestare il clock sotto controllo di programma.
3) Usare un clock ad alta frequenza cosicchè un errore di un periodo di clock è molto piccolo.
4) Allineare la CPU ed il clock attendendo un gradino di clock ovvero interrupt prima dell'inizio del conteggio.

PRIORITÀ DEL CLOCK IN TEMPO REALE

Un Interrupt da clock in tempo reale ha una priorità molto elevata, infatti la precisione degli intervalli di timing sarà influenzata da qualunque ritardo nel servizio. È pratica comune assegnare al clock in tempo reale la priorità più elevata tranne che l'insufficienza di potenza. La routine di servizio interrupt dal clok è generalmente realizzata estremamente corta in modo da non interferire con le altre attività della CPU.

a) Attesa per il Clock in Tempo Reale

Programma Sorgente:

```
RESET   EQU     0
        ORG     RESET
        LXI     SP,100H   ;PONE LO STACK ALLA FINE DELLA MEMORIA
        MVI     C,2       ;NUMERO DI INTERRUPT = 1
        DCR     C         ;PONE AD UNO 1 I FLAG
        EI                ;ABILITA INTERRUPT
WAITC:  JNZ     WAITC     ;ATTENDE PER INTERRUPT DA CLOCK
HERE:   JMP     HERE
;
        ORG     INTRP
        IN      PORT      ;AZZERA INTERRUPT
        DCR     C         ;CONTEGGIO INTERRUPT
        EI                ;RIABILITA INTERRUPT
        RET
```

Programma Oggetto:

Indirizzo di Mem. (Esadec.)	Istruzione (Mnemonico)			Contenuti di Mem. (Esadec.)
00	RESET:			
		LXI	SP,100H	31
01				00
02				01
03		MVI	C,2	0E
04				02
05		DCR	C	0D
06		EI		FB
07	WAITC:			
		JNZ	WAITC	C2
08				07
09				00
0A	HERE:	JMP	HERE	C3
0B				0A
0C				00
INTRP		IN	PORT	DB
+1				PORT
+2		DCR	C	0D
+3		EI		FB
+4		RET		C9

La routine di servizio interrupt disattiva il flip-flop di richiesta servizio (con IN PORT), conteggia e riabilita il sistema di interrupt. Naturalmente la routine potrebbe conteggiare progressivamente anzichè regredire e potrebbe comprendere riporti se i conteggi fossero troppo larghi per un singolo registro. Generalmente alcune locazioni RAM vengono riservate per il clock in tempo reale. Si noti che A non può essere usato come contatore poichè esso è impiegato dall'istruzione IN PORT che disattiva l'interrupt. Con il clock in tempo reale, la CPU non deve eseguire cicli di timing oppure attivare un timer. Il software è estremamente semplice e si può ottenere qualunque intervallo di timing desiderato.

b) Attesa per 10 Interrupt da Clock in Tempo Reale

Programma Sorgente:

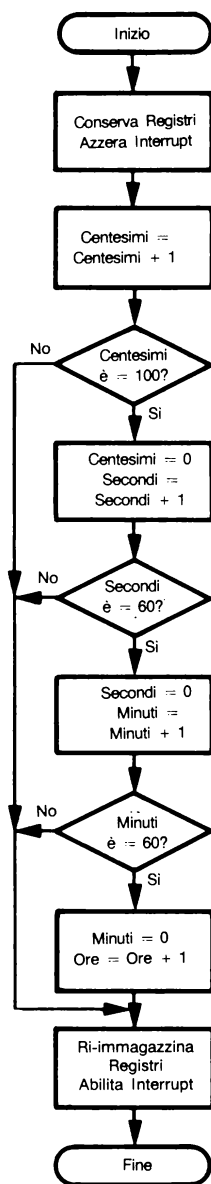
```
RESET   EQU   0
        ORG   RESET
        LXI   SP,100H  ;PONE LO STACK ALLA FINE DELLA MEMORIA
        MVI   C,11     ;NUMERO DI INTERRUPT = 11
        DCR   C        ;PONE AD UNO I FLAG
        EI             ;ABILITA INTERRUPT
WAITC:  JNZ   WAITC    ;ATTENDE PER 10 INTERRUPT
HERE:   JMP   HERE
        ORG   INTRP
        IN    PORT     ;AZZERA INTERRUPT
        DCR   C        ;CONTEGGIO INTERRUPT
        EI             ;RIABILITA INTERRUPT
        RET
```

Solo la variazione è il nuovo valore di partenza del Registro C. Si noti che si deve azzerare il flag Zero prima che la CPU esegua per la prima volta JNZ WAITC.

CONSERVAZIONE DEL TEMPO REALE

Una routine di interrupt da clock in tempo reale può conservare il tempo come segue nella memoria a partire dall'indirizzo RTCLK:

RTCLK — centesimi di secondo
RTCLK+1 — secondi
RTCLK+2 — minuti
RTCLK+3 — ore

Diagramma di Flusso:

La routine di servizio interrupt potrebbe essere:

```
        ORG     INTRP
        PUSH    PSW     ;CONSERVA REGISTRI
        PUSH    H
        IN      PORT    ;AZZERA INTERRUPT DA CLOCK IN TEMPO REALE
        LXI     H,RTCLK
        INR     M       ;AGGIORNA CENTESIMI DI SECONDO
        MVI     A,100
        CMP     M       ;C'È UN RIPORTO AI SECONDI?
        JNZ     DONE    ;NO, FATTO
        MVI     M,0     ;SÌ, CENTESIMI = 0
        INX     H
        INR     M       ;AGGIORNA SECONDI
        MVI     A,60
        CMP     M       ;C'È UN RIPORTO AI MINUTI?
        JNZ     DONE    ;NO, FATTO
        MVI     M,0     ;SÌ, SECONDI = 0
        INX     H
        INR     M       ;AGGIORNA MINUTI
        CMP     M       ;C'È UN RIPORTO ALLE ORE?
        JNZ     DONE    ;NO, FATTO
        MVI     M,0     ;SÌ, MINUTI = 0
        INX     H
        INR     M       ;AGGIORNA ORE
DONE:   EI              ;RIABILITA INTERRUPT
        POP     H       ;RI-IMMAGAZZINA I REGISTRI
        POP     PSW
        RET
```

Il programma principale potrebbe fornire un'attesa di 300 ms come segue:

```
        LXI     H,RTCLK
        MOV     A,M     ;ACCETTA IL TEMPO ATTUALE (CENTESIMI DI SECONDO)
        ADI     30      ;IL TEMPO RICHIESTO È DOPO 30 CONTEGGI
        CPI     100     ;MOD 100
        JC      WT30
        SUI     100
WT30:   CMP     M       ;ATTENDI FINO AL TEMPO RICHIESTO
        JNZ     WT30
```

Naturalmente il programma potrebbe eseguire altri compiti e controllare il tempo trascorso solo occasionalmente. Come si potrebbe produrre un ritardo di sette secondi? E tre minuti?

Un Interrupt da Telescrivente

INTERRUPT DA TELESCRIVENTE

Scopo: Il processore attende i dati che devono essere ricevuti da una magazzina gli stessi nella locazione di memoria 60.

UART INTERRUPT

1) Impiegando un UART

Il contenuto del programma è lo stesso di quello dell'interrupt da tastiera. Un UART compatibile con l'8080 come l'interfaccia di Comunicazione Programmabile 8251 è dotato delle uscite TRANSMITTER READY e RECEIVER READY che possono essere collegate agli ingressi interrupt dell'8080 ed 8085.
Questi bit sono automaticamente azzerati quando la CPU trasferisce dati al e dall'UART.

BIT DI INIZIO DELL'INTERRUPT

2) Impiegando un dispositivo 8212.
   Una routine di ricezione interrupt guidato deve avere i dati ricevuti sia all'ingresso dati dell'8212 ed anche, attraverso un invertitore, all'ingresso STROBE dell'8212. L'invertitore è necessario poichè il bit di partenza fornisce una transizione da alto a basso mentre l'8212 riconosce solo le transizioni del livello basso a quello alto.

Programma Sorgente:

```
RESET   EQU     0
        ORG     RESET
        LXI     SP,100H  ;PONI LO STACK ALLA FINE DELLA MEMORIA
        EI               ;ABILITA INTERRUPT
HERE:   JMP     HERE     ;ATTENDI DA TTY

        ORG     INTRP
        IN      PORT     ;AZZERA IL BIT DI INIZIO INTERRUPT
        CALL    TTYRCV   ;ACCETTA CARATTERE DALLA TTY
        STA     60H      ;SALVA CARATTERE
        EI               ;RIABILITA INTERRUPT
        RET
```

La subroutine TTYRCV è la subroutine di ricezione TTY mostrata al capitolo precedente senza la routine di riconoscimento del bit d'inizio.

Il gradino impiegato per causare l'interrupt è molto importante in questo caso. La transizione dal MARK normale o stato '1' allo SPACE o stato '0' deve causare un interrupt, poichè questa transizione identifica l'inizio della trasmissione di caratteri. La transizione da '0' ad '1' non si verificherà fino a che non viene ricevuto il bit di dati diversi da zero.

La routine di servizio deve disattivare il bit di inizio interrupt e lasciare gli interrupt disabilitati fino a che non è stato ricevuto un intero carattere. Diversamente ogni transizione da '1' a '0' nel carattere causerà un interrupt. Si noti che le transizioni ignorate azzereranno il flip-flop di richiesta servizio dell'8212. Comunque, leggendo i bit dati dalla porta si porrà ad uno il flip-flop senza causare un interrupt. È necessario riabilitare gli interrupt dopo che è stato letto un intero carattere.

ROUTINE DI SERVIZIO PIÙ GENERALE

COMPITI PER ROUTINE DI SERVIZIO GENERALE

Le routine di servizio generale, che costituiscono una parte di un sistema completo di guida interrupt, devono manipolare i compiti seguenti:

1) Conservare tutti i registri nello Stack cosicchè il programma interrotto può essere ripristinato correttamente.
   Si ricordi che l'istruzione Push dell'8080 trasferisce una coppia di registri nello Stack. PUSH PSW (PSW è la Parola di Stato del Processore) trasferisce quindi l'Accumulatore ed i flags nello Stack.
2) Impostazione della priorità di interrupt, probabilmente mediante scrittura del complemento della priorità nel registro di stato dell'8214.
   Il resto del sistema di interrupt può quindi essere riabilitato. Si ricordi che, allo scopo di immagazzinare correttamente la vecchia priorità, si deve conservarla nello Stack. Una copia della priorità attuale deve essere mantenuta in RAM, poichè il registro di Stato dell'8214 è di sola scrittura.
3) Ri-immagazzinamento della vecchia priorità, tutti i registri, e lo stato del sistema di interrupt prima del ritorno al programma interrotto.

La routine di servizio dovrebbe essere trasparente per quanto concerne il programma interrotto cioè essa non dovrebbe avere effetti incidentali.

Qualsiasi subroutine standard, impiegata da una routine di servizio interrupt deve essere rientrante. Se qualche subroutine non può essere rientrante, la routine di servizio interrupt deve avere una versione separata da usare.

I sistemi a priorità impieganti il Controllore 8214 possono manipolare più di otto sorgenti di interrupt.

Ogni punto di ingresso interrupt, eccetto quello a priorità più bassa, deve contenere un salto alla routine di servizio effettiva poichè sono disponibili solo alcune locazioni di memoria senza finire nel successivo punto d'ingresso.

IMPIEGO DI RST O

Se viene impiegato l'interrupt a priorità più elevata il sistema deve in qualche modo poter differenziare questo ingresso da RESET. Un flip-flop di stato (porta STAT), che conserva l'interrupt di livello più elevato (cioè $\overline{INT}$ dalla porta di priorità più elevata dell'8212), eseguirà il lavoro. Il programma è il seguente:

```
ORG     RESET
JMP     INTO      ;INGRESSO PER RESET OPPURE INTERRUPT
                  ;PIÙ ELEVATO
ORG     INTO
STA     TEMP      ;CONSERVA A IN CASO DI INTERRUPT
IN      STAT      ;L'INGRESSO È CAUSATO DA INTERRUPT?
ORA     A
JNZ     START     ;NO, VA ALLA ROUTINE RESET
LDA     TEMP      ;SÌ, RI-IMMAGAZZINA A

        ROUTINE DI SERVIZIO A PRIORITÀ PIÙ ELEVATA
```

Si noti A non può essere conservato nello Stack poichè RESET non inizializza il Puntatore dello Stack. L'alternativa è di non impiegare il livello di interrupt più elevato.

In generale occorre fare attenzione quando un interrupt può disturbare un programma. Ovviamente i cicli di timing e le altre routine con precise funzioni di timing non possono essere interrotti. Così pure le routine startup oppure le routine che inizializzano i parametri per le routine di servizio. La routine di servizio deve non solo conservare i contenuti dei registri e flag ma essa deve anche non interferire con operazioni eseguite parzialmente. Per esempio, si noti che un trasferimento a oppure da un buffer ad ingresso/uscita guidato da interrupt può non essere interrotto. Diversamente la routine d'uscita può essere interrotta ed avere il byte dei dati da trasferire sostituito da un nuovo byte proveniente dalla routine d'ingresso. Occorre essere particolarmente cauti quando si sta manipolando dati di diverse parole. Un interrupt può influire sui dati e sul risultato quando una parte di quest'ultimo viene aggiornata ed una no. Queste mescole possono creare difficoltà notevoli nel debugging (collaudo). Le routine di servizio più generale devono conservare tutti i registri e la vecchia priorità nello Stack all'ingresso e ri-immagazzinati prima del ritorno. Le procedure standard sono:

CONSERVAZIONE E RI-IMMAGAZZINAMENTO DEI REGISTRI E DELLA PRIORITÀ

CONSERVAZIONE REGISTRI E PRIORITÀ

```
        PUSH    PSW         ;CONSERVA I REGISTRI
        PUSH    B
        PUSH    D
        PUSH    H
        LDA     PRTY        ;CONSERVA LA VECCHIA PRIORITÀ
        PUSH    PSW
        MVI     A,PRTYN     ;PONE LA NUOVA PRIORITÀ
        STA     PRTY
        OUT     PRREG
        EI
```

RI-IMMAGAZZINAMENTO DEI REGISTRI E DELLA PRIORITÀ

```
        POP     PSW
        STA     PRTY        ;RI-IMMAGAZZINA LA VECCHIA PRIORITÀ
        OUT     PRREG
        POP     H           ;RI-IMMAGAZZINA I REGISTRI
        POP     D
        POP     B
        POP     PSW
```

Se è impiegato un dispositivo 8214 è necessaria una copia della vecchia priorità per più di due livelli poichè il registro di priorità dell'8214 è di sola scrittura ed il ritorno deve ripristinare il livello corretto.

Naturalmente la vecchia priorità ed il più basso livello non devono essere conservati e ri-immagazzinati poichè essi sono i più bassi possibile. Si noti che il programma deve riscrivere la vecchia priorità nel registro di stato dell'8214 prima del ritorno.

PROBLEMI

1) Un Interrupt di Test

Scopo: Il computer attende che si verifichi un interrupt e quindi esegue l'istruzione di test:

```
HERE:   JMP     HERE
```

fino a che non interviene un successivo interrupt.

2) Un Interrupt da Tastiera

Scopo: Il computer attende un ingresso di quattro digit da una tastiera e posiziona i dati nelle locazioni di memoria da 60 a 63 (quello ricevuto prima in 60). Ogni ingresso di un digit causa un interrupt. Il quarto ingresso potrebbe anche risolversi nella disabilitazione del sistema di interrupt.

Problema Campione:

Dati dalla Tastiera: = 04, 06, 01, 07
Risultato = (60) = 04
(61) = 06
(62) = 01
(63) = 07

3) Un Interrupt da Stampante:

Scopo: Il computer invia i quattro caratteri delle locazioni di memoria da 60 a 63 (prima 60) alla stampante. Ogni carattere è richiesto da interrupt. Il quarto trasferimento disabilita anche il sistema di interrupt.

4) Un Interrupt da Clock in Tempo Reale

Scopo: Il computer azzera la locazione di memoria 60 per inizializzare e quindi attende un interrupt da Clock in tempo reale. Ogni volta che ricorre un interrupt in tempo reale, il programma complementa la locazione di memoria 60. Come si potrebbe modificare il programma in modo che esso complementi la locazione di memoria ogni 10 interrupt? Come si potrebbe modificare il programma in modo che la locazione di memoria 60 sia zero per dieci periodi di clock, FF (esadecimale) per cinque periodi di clock e così via continuamente? Si può usare un display piuttosto che la locazione di memoria 60 cosicchè sarà più facile da vedere.

5) Un Interrupt da Telescrivente

Scopo: Il computer riceve dati TTY da un'interruzione UART ed immagazzina i caratteri in un buffer iniziante nella locazione di memoria 60. Quando il computer riceve un carattere di ritorno carrello (OD esadec.) disabilita il sistema di interrupt ed arresta la ricezione dei caratteri. Come si potrebbe cambiare il programma impiegando una porta di I/O 8212? Si assuma che la subroutine TTYRCV sia disponibile come nell'esempio.

Capitolo 13

DEFINIZIONE DEL PROBLEMA E PROGETTO DEL PROGRAMMA

I COMPITI DELLO SVILUPPO DEL SOFTWARE

Nei capitoli precedenti è stata concentrata l'attenzione nella scrittura di brevi programmi in linguaggio assembly. Sebbene questo argomento sia importante esso è solo una piccola parte del problema globale dello sviluppo del software. Benchè la scrittura di programmi in linguaggio assembly sembri il compito principale per i principianti, esso talvolta diventa relativamente semplice. Fino da ora si dovrebbe avere familiarità con la parte dei metodi standard per la programmazione in linguaggio assembly sul microprocessore 8080. I quattro capitoli successivi descriveranno come trasferire i compiti in programmi e come mettere assieme programmi brevi per formare un sistema di lavoro.

STADI DELLO SVILUPPO DEL SOFTWARE

Lo sviluppo del software consiste di molti stadi. La Figura 13-1 è un diagramma di flusso del processo di sviluppo del software. I suoi stadi sono:

- Definizione del problema
- Progetto del programma
- Codifica
- Debugging
- Testing
- Documentazione
- Manutenzione e riprogetto

Ognuno di questi è importante nell'introduzione di un sistema di lavoro. Si noti che la codifica, cioè la scrittura del programma in forma comprensibile al computer, è uno solo dei sette stadi.

IMPORTANZA RELATIVA DELLA CODIFICA

Di fatto è lo stadio più facile da definire e da eseguire. Le regole della scrittura dei programmi del computer sono relativamente facili da imparare. Esse variano talvolta da computer a computer ma le tecniche di base rimangono le stesse; solo pochi progetti software operano in modo preoccupante a causa dello stadio di codifica; infatti lo stadio di codifica è la parte più significativa dello sviluppo del software. Gli esperti stimano che un programmatore può scrivere da uno a dieci statement pienamente collaudati e documentati. Chiaramente la sola codifica di uno o dieci statement è difficilmente uno sforzo pieno. Nella maggior parte dei progetti software la codifica occupa meno del 25% del tempo del programmatore.

VALUTAZIONE DELL'AVANZAMENTO NEGLI STADI

La valutazione dell'avanzamento degli altri stadi è difficoltosa. Si può dire che la metà degli errori è stata rimossa o metà del problema è stato definito. La conclusione di stadi come progetto del programma, debugging e testing è difficile da stabilire. Molti giorni o settimane di sforzi possono risultare in progressi non ben definiti. Inoltre ùn lavoro incompleto in uno stadio può risolversi in gravi problemi successivi. Per esempio una scarsa definizione del problema o progetto del programma possono rendere molto difficoltosi il debugging ed il testing. Il tempo risparmiato in uno stadio può essere consumato molte volte in uno stadio successivo.

Figura 13-1
Diagramma Di Flusso Dello Sviluppo Software

DEFINIZIONE DEGLI STADI

DEFINIZIONE DEL PROBLEMA

La definizione del problema è la formulazione del compito nei termini delle specifiche posizionate sul computer. Per esempio, cosa è necessario fare per controllare un utensile col computer, eseguire una serie di test elettrici ovvero manipolare comunicazioni tra una centrale regolatrice ed uno strumento remoto? La definizione del problema richiede la determinazione della forma e della velocità degli ingressi e delle uscite, la quantità e velocità di elaborazione richieste ed i tipi di errore possibili e la loro manipolazione. La definizione del problema richiede un'idea vagamente determinata della costituzione di un sistema controllato da computer e definisce i compiti e le richieste per il computer.

PROGETTO DEL PROGRAMMA

Il progetto del programma è una caratteristica del programma del computer che eseguirà i compiti che sono definiti. Nel progetto i compiti sono descritti in un modo che può essere facilmente convertito in un programma. Tra le tecniche più comuni in questo stadio si segnalano la costruzione del diagramma di flusso, la programmazione strutturata, la programmazione modulare e progetto top-down.

CODIFICA

La codifica è la scrittura del programma in una forma che il computer può capire direttamente o tradurre. La forma suddetta può essere linguaggio di macchina, linguaggio assembly oppure linguaggio ad alto livello.

DEBUGGING

Il debugging, detto anche verifica del programma permette di rendersi conto se il programma esegue quanto richiesto dalle specifiche di progetto. In questo stadio si usano strumenti come breakpoint, trace, simulatori, analizzatori logici ed emulatori circuitali. La conclusione dello stadio di debugging è difficile da definire poichè non si saprà mai quando si è trovato l'ultimo errore.

TESTING

Il testing, definito anche una convalida del programma, assicura che il programma esegua correttamente i compiti di sistema globali. Il progettista impiega simulatori, utilizzatori e varie tecniche statistiche per eseguire alcune misure dell'esecuzione del programma.

DOCUMENTAZIONE

La documentazione è la descrizione del programma in forma propria per utenti e per il personale della manutenzione. La documentazione permette anche al progettista di sviluppare una biblioteca di programmi in modo che i compiti successivi siano molto più semplici. I diagrammi di flusso, i commenti, le mappe di memoria e le forme di biblioteca sono alcuni degli strumenti impiegati nella documentazione.

MANUTENZIONE E RIPROGETTO

La manutenzione ed il riprogetto sono la revisione, il miglioramento e l'estensione del programma. Chiaramente il progettista deve essere pronto per manipolare problemi di settore in attrezzature basate sul computer. Possono essere richiesti dei metodi speciali di diagnosi ed altri strumenti di manutenzione. Il miglioramento o l'estensione del programma possono essere necessari per fare in modo che il programma soddisfi a nuove richieste oppure manipoli nuovi compiti.
Il resto di questo capitolo considererà solo gli stadi di definizione del problema ed il progetto del programma. Il Capitolo 14 discuterà il debugging ed il testing ed il Capitolo 15 discuterà la documentazione, estensione e riprogetto.
Si considereranno tutti gli stadi assieme, in alcuni semplici esempi al Capitolo 16.

DEFINIZIONE DEL PROBLEMA

I compiti tipici del microprocessore richiedono una quantità di definizione. Per esempio cosa deve fare un programma per controllare una scala, un registro di cassa oppure un generatore di segnali? Chiaramente c'è una lunga strada per attivare a definire i compiti coinvolti.

DEFINIZIONE DEGLI INGRESSI

Come si può iniziare la definizione? La posizione ovvia per cominciare è con gli ingressi. Si comincerà elencando tutti gli ingressi che il computer può ricevere in questa applicazione.

Esempi di ingressi sono:

- Blocchi di dati da linee di trasmissione
- Parole di stato da periferiche
- Dati da Convertitori A/D

Quindi ci si può chiedere quanto segue per ogni ingresso:

1) Quale è la forma, cioè quali segnali riceverà effettivamente il computer?
2) Quando è disponibile l'ingresso e come fa il processore a conoscere se è disponibile? Il processore può richiedere l'ingresso con un segnale strobe? L'ingresso può fornire il proprio clock?
3) Per quanto tempo è disponibile?
4) Con che frequenza cambia e come può conoscere il processore se è cambiato?
5) Fornisce una sequenza o blocco di dati? L'ordine è importante?
6) Cosa succederebbe se i dati contenessero errori? Questi possono comprendere errori di trasmissione, dati non corretti, errori di sequenza, dati ulteriori, ecc..
7) Questo ingresso è funzione degli altri ingressi o uscite?

DEFINIZIONE DELLE USCITE

La fase successiva da definire è costituita dalle uscite. Si comincerà elencando tutte le uscite che il computer deve produrre. Tra gli esempi di uscite si ricordano:

- Blocchi di Dati a linee di trasmissione.
- Parole di Controllo a periferiche.
- Dati a Convertitori D/A.

Quindi ci si può chiedere quanto segue per ogni uscita:

1) Qual è la sua forma, cioè quali segnali deve produrre il computer?
2) Quando deve essere disponibile e come può conoscere la periferica se è disponibile?
3) Per quanto tempo deve essere disponibile?
4) Con quale frequenza cambia e come può conoscere la periferica se è cambiata?
5) C'è una sequenza di uscite? L'ordine è importante?
6) Cosa si potrebbe fare per evitare errori di trasmissione o per rivelare o recuperare insufficienze delle periferiche?
7) Questa uscita come è collegata agli altri ingressi ed uscite?

SEZIONE DI ELABORAZIONE

Tra la lettura dei dati di ingresso e la fornitura dei risultati di uscita si colloca la sezione di elaborazione. Qui si deve esattamente determinare come il computer deve elaborare i dati di ingresso. Le domande sono:

FATTORI NELL'ELABORAZIONE

1) Qual è la procedura di base (algoritmo) per la trasformazione dei dati di ingresso in risultati di uscita?
2) Quali limiti di tempo esistono? Questi possono comprendere velocità dati, tempi di ritardo, le costanti di tempo dei dispositivi di ingresso ed uscita, ecc..

3) Quali limiti di memoria esistono? Ci sono limiti quantitativi della memoria di programma oppure memoria dati oppure nella dimensione dei buffer?
4) Quali tabelle o programmi standard devono essere impiegati? Quali sono le loro esigenze?
5) Quali casi speciali esistono e come potrebbe manipolarli il programma?
6) Quanto deve essere preciso il risultato?

MANIPOLAZIONE DEGLI ERRORI

Un fattore importante in molte applicazioni è la manipolazione di errori. Chiaramente il progettista deve fare previsioni per rimediare errori comuni e per l'individualizzazione di malfunzionamenti. Tra i problemi che il progettista deve porsi allo stadio di definizione ci sono:

CONSIDERAZIONI SUGLI ERRORI

1) Quali errori potrebbero verificarsi?
2) Quali errori sono molto probabili? Se una persona fa operare il sistema, l'errore umano è il più comune. Dopo gli errori umani, gli errori di comunicazione o trasmissione sono più comuni degli errori meccanici, elettrici o del processore.
3) Quali errori non saranno immediatamente ovvi al sistema? Un problema speciale è la ricorrenza di errori che il sistema o l'operatore non possono rivelarli immediatamente.
4) Come può il sistema rimediare ad errori con la minima perdita di tempo e dati ed ancora essere consapevole che si è verificato un errore?
5) Quali errori o malfunzionamenti originano lo stesso comportamento del sistema? Come questi errori o malfunzionamenti possono essere separati per scopi diagnostici?
6) Quali errori coinvolgono procedure di sistema speciali? Per esempio l'errore di parità richiede la ri-trasmissione dei dati?

Un altro problema è: Come può il tecnico di settore trovare sistematicamente la sorgente dei malfunzionamenti senza essere un esperto? Possono essere di aiuto programmi di test built-in (incorporati) (vedere per esempio V.P. Srini, «Fault Diagnosis of Microprocessor Systems», Computer, Gennaio 1977, pp. 60-65), diagnostici speciali ed analisi di sigle. Per una descrizione dell'analisi di sigle, vedere Gordon, G. ed H. Nedig, «Hexadecimal Signatures Identify Trouble-spots in Microprocessor Systems», Electronics, 3 Marzo 1977, pp. 89-96. Esiste anche una nota applicativa (#222) intitolata «A Designer's Guide to Sigmature Analysis» disponibile presso la Hewlett-Packard.

FATTORI UMANI

INTERAZIONE CON L'OPERATORE

Molti sistemi basati su microprocessori coinvolgono l'interazione umana. I fattori umani devono essere considerati attraverso il processo di sviluppo di tali sistemi. Tra le domande che il progettista deve chiedersi ci sono:

1) Quali procedure d'ingresso sono le più naturali per l'operatore umano?
2) Come conoscerà un operatore dove l'operazione d'ingresso inizia, continua e termina?
3) Come può essere informato l'operatore degli errori procedurali e dei malfunzionamenti dell'attrezzatura?
4) Quali errori può compiere l'operatore con probabilità maggiore?
5) Come può conoscere l'operatore quali dati sono entrati correttamente?
6) I display sono in una forma che l'operatore può leggere e capire facilmente?
7) La risposta del sistema è adeguata per l'operatore?
8) È facile per l'operatore usare il sistema?
9) Ci sono delle strutture guidanti per un operatore inesperto?
10) Ci sono delle scorciatoie e delle scelte ragionevoli per un operatore esperto?

La costruzione di un sistema di massa è difficoltosa. Il microprocessore può rendere il sistema più potente, più flessibile e più sensibile. Comunque il progettista deve ancora aggiungere un tocco umano che può aumentare la praticità e l'attraenza del sistema e la produttività dell'operatore umano.

ESEMPI

Risposta ad un Interruttore

DEFINIZIONE DI UN SISTEMA INTERRUTTORE DISPLAY LUMINOSO

La Figura 13-2 mostra semplice sistema il cui ingresso è un interruttore SPST (unipolare ad una via) e l'uscita è un singolo display LED. In risposta alla chiusura dell'interruttore, il processore accende il display per un secondo. Sicuramente questo sistema sarebbe facile da definire.

INGRESSO INTERRUTTORE E DISPLAY LUMINOSO

Verrà prima esaminato l'ingresso e la risposta ad ognuna delle domande precedentemente presentate.

1) La forma dell'ingresso è un singolo bit che può essere '0' (interruttore chiuso) oppure '1' (interruttore aperto).
2) L'ingresso è sempre disponibile e non necessita di essere richiesto.
3) L'ingresso è disponibile per almeno diversi millisecondi dopo la chiusura.
4) L'ingresso raramente cambierà più di una volta ogni qualche secondo. Il processore deve soltanto manipolare il rimbalzo dell'interruttore. Il processore deve ispezionare l'interruttore per determinare quando esso è chiuso.
5) Non esiste una sequenza di ingressi.
6) Gli errori più ovvi dell'ingresso sono il guasto dell'interruttore, guasto nella circuiteria dell'ingresso, ed i tentativi dell'operatore volti a chiudere l'interruttore prima che sia trascorso un certo tempo. Verrà discussa in seguito la manipolazione di questi errori.
7) L'ingresso non dipende da nessun altro ingresso od uscita.

L'esigenza successiva nella definizione del sistema è di esaminare l'uscita. La risposta alle corrispondenti domande sono:

USCITE INTERRUTTORE E DISPLAY LUMINOSO

1) La forma dell'uscita è un singolo bit che è '0' per accendere il display, '1' per spegnerlo.
2) Non ci sono limiti di tempo sull'uscita. La periferica non richiede di essere informata sulla disponibilità di dati.
3) Se il display è un LED i dati devono essere disponibili per alcuni millisecondi alla velocità di impulso di circa 100 volte al secondo. L'osservatore vedrà il display continuamente illuminato.
4) I dati devono cambiare dopo un secondo cioè diventare off.
5) Non c'è una sequenza di uscite.
6) I possibili errori di uscita sono il guasto del display ed il guasto della circuiteria d'uscita.
7) L'uscita dipende dall'interruttore d'ingresso e dal tempo.

L'ingresso dell'interruttore è un '1' se l'interruttore è aperto ed uno '0' se chiuso. La CPU applica l'uscita al catodo del LED; uno '0' accende il display.

Figura 13-2
Il Sistema Interruttore - Display

La sezione di elaborazione è estremamente semplice. Non appena l'interruttore d'ingresso diventa uno '0' logico, la CPU accende il display ('0' logico) per un secondo. Non esistono limiti di tempo o memoria.

MANIPOLAZIONE DI ERRORI DELL'INTERRUTTORE E DEL DISPLAY

Consideriamo ora gli errori ed i malfunzionamenti possibili. Questi sono:

1) Un'altra chiusura dell'interruttore prima che sia trascorso un secondo.
2) Guasto dell'interruttore.
3) Guasto del display.
4) Guasto del computer.

Sicuramente il primo errore è il più probabile. La soluzione più semplice per il processore è di ignorare chiusure dell'interruttore fino a che non è trascorso un secondo. Questo breve periodo di non disponibilità sarà difficilmente notato da un operatore umano. Inoltre non considerando l'interruttore durante questo periodo significa che non è necessaria circuiteria o software di eliminazione rimbalzo poichè il sistema non può reagire in alcun modo al rimbalzo dell'interruttore.

Chiaramente gli ultimi tre guasti possono produrre risultati impredicibili. Il display può permanere acceso, spento o variare stato a caso. Alcuni modi possibili per isolare i guasti possono essere:

1) Disporre di hardware per il lamp-test per controllare il display cioè un pulsante che accenda il display indipendentemente dal processore.
2) Una connessione diretta dell'interruttore per controllare il suo funzionamento.
3) Un programma diagnostico per verificare i circuiti d'ingresso e d'uscita.

Se il display e l'interruttore sono entrambi funzionanti è guasto il computer. Un tecnico di settore con attrezzature proprie può quindi determinare la causa del guasto.

Un Caricatore di Memoria Basato sugli Interruttori

DEFINIZIONE DI UN CARICATORE DI MEMORIA BASATO SUGLI INTERRUTTORI

La Figura 13-3 mostra un sistema che permette all'utente di fare entrare dei dati in una locazione di memoria di un microcomputer. Una porta d'ingresso DPORT, legge i dati da otto interruttori a ginocchiera (toggle). Un'altra porta di ingresso, CPORT, viene impiegata per leggere le informazioni di controllo. Queste sono costituite da tre interruttori istantanei, indirizzo elevato, indirizzo basso, dati. L'uscita è il valore dell'ultimo ingresso completo dagli interruttori dei dati; per il display sono impiegati 8 display LED.

Il sistema, naturalmente, richiede anche vari resistori, buffer e driver.

Verranno esaminati prima gli ingressi. Le caratteristiche degli interruttori sono le stesse dell'esempio precedente; comunque c'è una distinta sequenza di ingressi come segue:

1) L'operatore deve porre gli interruttori dei dati in accordo con gli otto bit più significativi di un indirizzo, quindi
2) premere il pulsante INDIRIZZO ALTO. I bit dell'indirizzo elevato appariranno sul display ed il programma interpreterà i dati come il byte più significativo dell'indirizzo.
3) Quindi l'operatore deve porre gli interruttori dei dati in accordo con il valore del byte meno significativo dell'indirizzo e
4) premere il pulsante INDIRIZZO BASSO. I bit dell'indirizzo basso appariranno sul display ed il programma interpreterà i dati come il byte meno significativo dell'indirizzo.

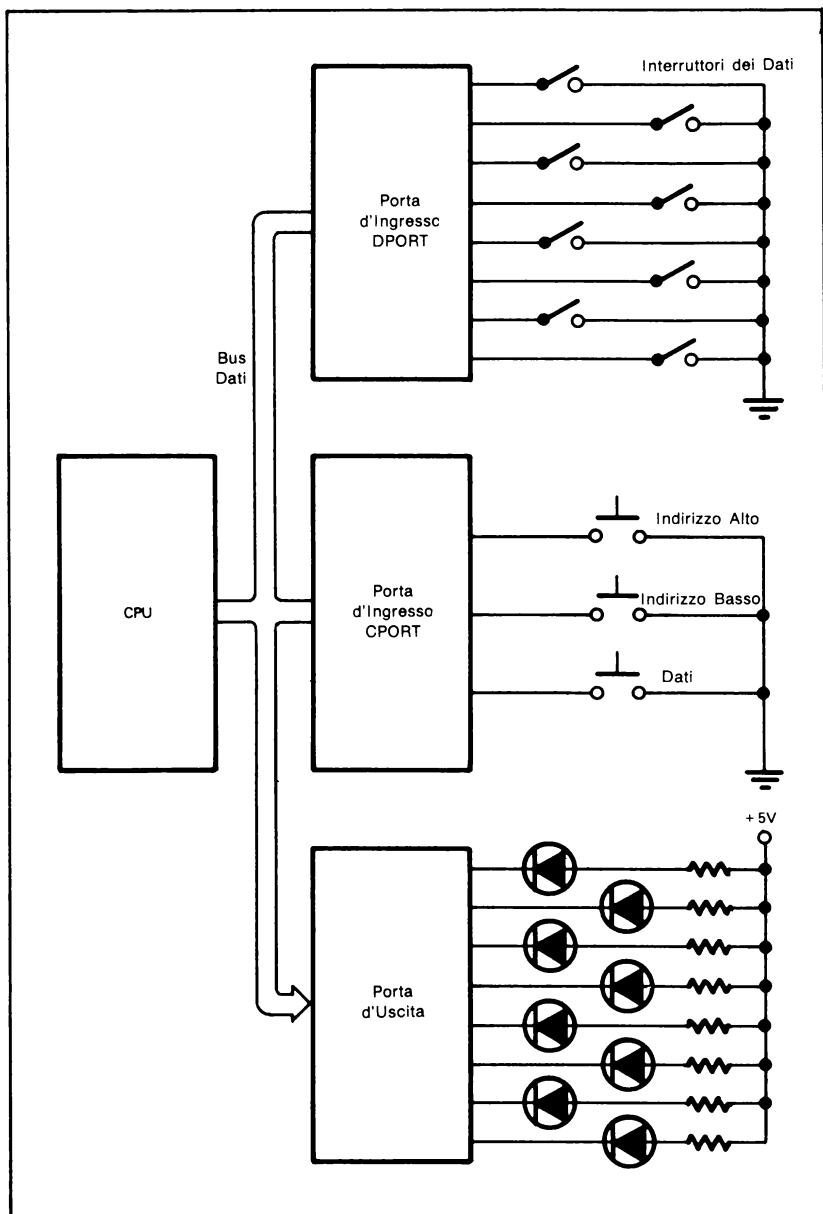



Figura 13-3
Il Caricatore Di Memoria Basato Sull'Interruttore

5) Infine l'operatore deve caricare i dati desiderati negli interruttori dei dati e
6) premere il pulsante DATI. Il display mostrerà ora i dati ed il programma immagazzinerà i dati in memoria all'indirizzo entrato precedentemente.

L'operatore può ripetere il processo per fare entrare un intero programma. Chiaramente, anche in questa situazione semplificata, ci sono molte sequenze possibili da considerare. Come ci si comporta con sequenze errate e come si semplifica l'impiego del sistema?
L'uscita non costituisce un problema. Dopo ogni ingresso il programma invia al display il complemento (poichè il display è attivo al livello basso) dei bit d'ingresso. I dati di uscita rimangono gli stessi fino alla successiva operazione di ingresso.

La sezione di elaborazione rimane abbastanza semplice. Non ci sono limiti di tempo o memoria. Il programma può eliminare il rimbalzo degli interruttori attendendo per alcuni millisecondi e deve fornire il complemento dei dati ai display.

ERRORI DI MANIPOLAZIONE DEL CARICATORE DI MEMORIA

Gli errori più comuni sono dovuti all'operatore. Questi comprendono:

1) Ingressi non corretti.
2) Ordine non corretto.
3) Ingressi incompleti, cioè dati dimenticati.

Il sistema deve essere in grado di manipolare questi problemi in un modo ragionevole in quanto essi certamente ricorreranno nel funzionamento effettivo.

Il progettista deve anche considerare gli effetti di guasto dell'attrezzatura. Come precedentemente le difficoltà possibili sono:

1) Guasto dell'interruttore.
2) Guasto del display.
3) Guasto del computer.

In questo sistema comunque, si deve fare più attenzione a come questi guasti influenzano il sistema. Un guasto del computer, presumibilmente, originerà un comportamento particolare del sistema e sarà facilmente rivelabile. Un guasto del display può non essere notato immediatamente; qui la possibilità di LAMP TEST permetterà all'operatore di controllare il funzionamento del display. Si noti che sarebbe auspicabile poter controllare ogni display separatamente in modo da rivelare il caso in cui le linee di uscita sono circuitate assieme. Inoltre l'operatore può non rivelare immediatamente un guasto dell'interruttore; comunque l'operatore dovrebbe notarlo quasi subito e stabilire l'interruttore guasto mediante un processo di eliminazione.

CORREZIONE DEGLI ERRORI DELL'OPERATORE NEL CARICATORE DI MEMORIA

Si considerano ora alcuni possibili errori dell'operatore. Errori tipici saranno:

1) Dati errati.
2) Ordine degli ingressi e degli interruttori errato.
3) Tentativi di passare all'ingresso successivo senza completare quello attuale.

L'operatore, presumibilmente, noterà i dati errati non appena compaiono sul display. Qual è una procedura di recupero praticabile per l'operatore? Alcune scelte possono essere:

1) L'operatore deve completare la procedura d'ingresso, cioè far entrare INDIRIZZO BASSO e DATI se l'errore ricorre in INDIRIZZO ALTO. Chiaramente questa procedura è tediosa e potrebbe solo servire ad annoiare l'operatore.
2) L'operatore può far ripartire il processo d'ingresso ritornando alle fasi di ingresso dell'indirizzo elevato. Questa soluzione è comoda se l'errore è in INDIRIZZO ALTO, ma forza l'operatore a far rientrare i dati precedenti se l'errore è nello stadio INDIRIZZO BASSO o DATI.

3) L'operatore può far entrare qualunque parte della sequenza in ogni istante semplicemente ponendo gli interruttori DATI con i dati desiderati e premendo il corrispondente pulsante. La procedura permette all'operatore di fare correzioni a qualunque punto della sequenza.

Questo tipo di procedura dovrebbe essere preferita rispetto a quella che non permette una correzione immediata dell'errore. Infatti quest'ultima procedura ha una certa varietà di fasi conclusive o dati di ingresso nel sistema che non consentono all'operatore un controllo finale. Qualunque complicazione aggiuntiva o software sarà giustificata da un aumento dell'efficienza dell'operatore. Si potrebbe sempre preferire il lasciare al microcomputer il lavoro tedioso ed ispezionare arbitrariamente le sequenze; infatti esso è instancabile e non tralascia nulla rispetto al manuale di funzionamento.

Un'altra caratteristica utile potrebbe essere un'esposizione dello stato per definire il significato del display. Tre elementi luminosi di stato indicati INDIRIZZO ALTO, INDIRIZZO BASSO e DATI permetterebbero all'operatore di conoscere cosa è entrato senza dover ricordare quale pulsante è stato premuto. Il processore avrebbe così un monitor di sequenza ma la complicazione aggiuntiva in software semplificherebbe i compiti dell'operatore.

Si potrebbe notare che, sebbene sia stata messa in evidenza l'interazione umana, l'interazione della macchina o del sistema ha molte delle stesse caratteristiche. Il microprocessore dovrebbe funzionare. Se la complicazione dei compiti del microprocessore rende semplice il recupero degli errori ed ovvie le cause di guasto, l'intero sistema lavorerà meglio e sarà più facile da mantenere. Si noti che non si possono stabilire considerazioni sull'uso del sistema e sulla manutenzione fino alla fine del processo di sviluppo software; mentre esse dovrebbero essere correttamente comprese nello stadio di definizione del problema.

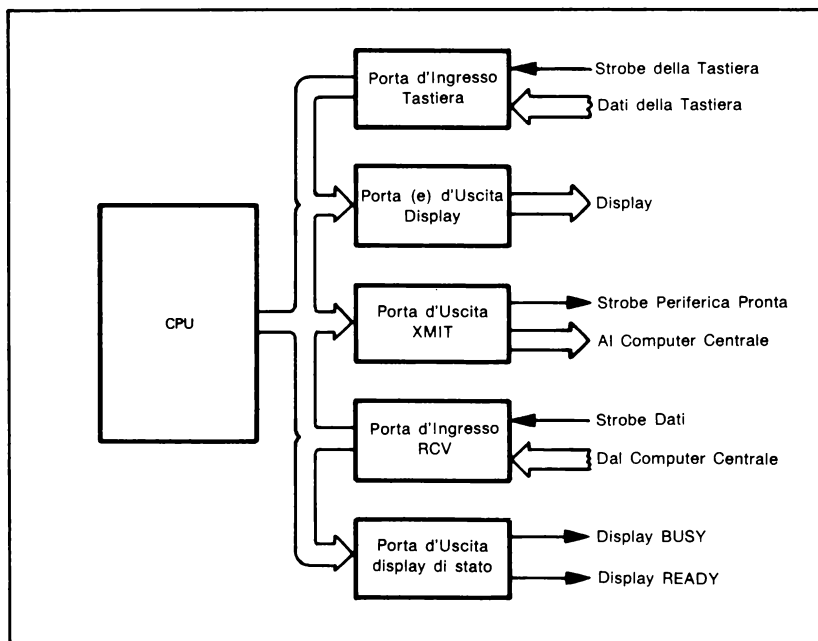



Figura 13-4
Diagramma A Blocchi Di Un Terminale Di Verifica

Un Terminale di Verifica

DEFINIZIONE DI UN TERMINALE DI VERIFICA

La Figura 13-4 è un diagramma a blocchi di un terminale di verifica del credito. Una porta d'ingresso deriva i dati da una tastiera (Vedere Figura 13-5); l'altra porta d'ingresso accetta i dati di verifica da una linea di trasmissione. Una porta d'uscita invia i dati ad un set di display (Vedere Figura 13-6); un'altra invia il numero della carta di credito al computer centrale. Una terza porta d'uscita commuta on una spia ogni volta che il terminale è pronto ad accettare un'indagine ed un'altra spia quando l'operatore invia l'informazione. La spia BUSY si spegne quando ritorna il responso. Chiaramente l'ingresso e l'uscita dei dati sarà più complessa del caso precedente benchè l'elaborazione sia ancora abbastanza semplice.

Figura 13-5
Tastiera Del Terminale Di Verifica

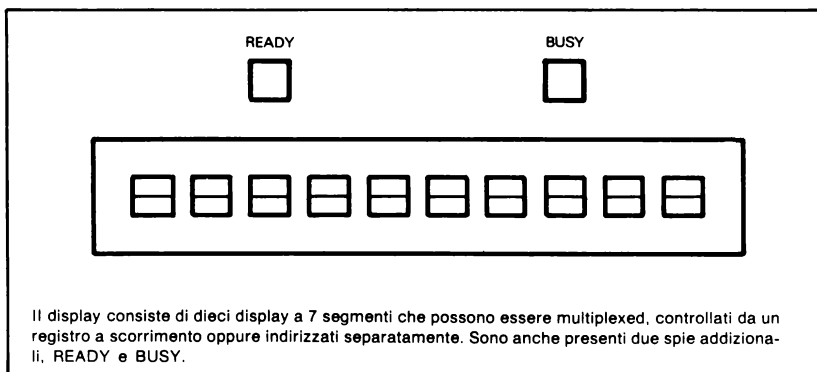



Figura 13-6
Display Del Terminale Di Verifica

Dei display addizionali possono essere comodi per sottolineare il significato della risposta. Molti terminali impiegano una spia verde per YES ed una spia rossa per NO ed una gialla per CONSULT STORE MANAGER. Si noti che queste spie devono essere chiaramente contrassegnate con i loro significati per considerare il caso di operatore daltonico.

INGRESSI DEL TERMINALE DI VERIFICA

Verrà prima considerato l'ingresso della tastiera. Questo è, naturalmente, diverso dall'ingresso dell'interruttore poichè la CPU deve avere modo di distinguere i nuovi dati. Si assumerà che ogni chiusura di tasto fornisca un unico codice esadecimale (si possono così codificare tutti i tasti in un digit poichè ci sono 12 tasti) ed uno strobe. Il programma dovrà riconoscere lo strobe ed andare a prendere il numero esadecimale che identifica il tasto. C'è un limite di tempo poichè il programma non può omettere nessun dato o strobe. Il limite non è preoccupante poichè gli ingressi della tastiera saranno separati almeno da diversi millisecondi.

Analogamente l'ingresso di trasmissione consiste di una serie di caratteri, ognuno identificato da uno strobe (forse da un UART). Il programma dovrà riconoscere ogni strobe ed andare a prendere il carattere. I dati, essendo inviati attraverso linee di trasmissione sono normalmente organizzati in messaggi. Un possibile formato di messaggio è:

1) Testa.
2) Indirizzo del Terminale di Destinazione.
3) SÌ oppure NO Codificato.
4) Coda.

Il terminale controllerà la testa, leggerà l'indirizzo di destinazione e vedrà se il messaggio gli è destinato. Se il messaggio è destinato al terminale, quest'ultimo accetta i dati. L'indirizzo potrebbe essere (e spesso lo è) hard-wired nel terminale cosicchè il terminale riceve solo messaggi destinatigli. Questo approccio semplifica il costo al prezzo di qualche caratteristica di flessibilità.

USCITE DEL TERMINALE DI VERIFICA

Anche l'uscita è più complessa dei casi precedenti. Se i display sono multiplexed il processore deve non solo inviare dati alla porta display ma anche dirigere i dati ad un particolare display. Per manipolare questo è necessario sia una porta di controllo separata sia un contatore e decodificatore. Si noti che i controlli di azzeramento possono generare gli zeri iniziali come pure il primo digit di un numero multi-digit che non è mai zero. Il software può anche manipolare questo compito. I limiti di tempo comprendono la lunghezza dell'impulso e la frequenza richiesta per produrre un display apparentemente continuo per l'operatore.

L'uscita delle comunicazioni consisterà di una serie di caratteri con un particolare formato. Il programma dovrà anche considerare il tempo richiesto tra i caratteri. Un formato possibile per il messaggio di uscita è:

1) Testa.
2) Indirizzo terminale.
3) Numero carta di credito.
4) Coda.

Un computer centrale delle comunicazioni può essere impiegato per raccogliere i terminali, controllare se i dati da inviare sono pronti.

L'elaborazione in questo sistema coinvolge molti compiti nuovi come:

1) Identificazione dei tasti di controllo mediante numero ed esecuzione di azioni appropriate.
2) Aggiunta della testa e della coda al messaggio uscente.
3) Riconoscimento della testa e coda nel messaggio di ritorno.
4) Controllo dell'indirizzo di terminale entrante.

MANIPOLAZIONE DELL'ERRORE NEL TERMINALE DI VERIFICA

Si noti che nessuno dei compiti coinvolge qualsiasi aritmetica complessa oppure qualunque limite preoccupante di tempo o memoria.

Il numero di errori possibili in questo sistema, naturalmente, è molto maggiore di quello degli esempi precedenti. Saranno prima considerati gli errori possibili per l'operatore. Questi comprendono:

1) Ingresso non corretto della carta di credito.
2) Tentativo di inviare un numero incompleto della carta di credito.
3) Tentativo di inviare un altro numero mentre il computer centrale se ne sta elaborando uno.
4) Azzeramento degli ingressi non esistenti.

Alcuni di questi errori possono essere facilmente manipolati mediante una strutturazione corretta del programma. Per esempio il programma non accetterebbe il tasto SEND fino a quando il numero della carta di credito non è entrato correttamente e lo stesso programma ignorerebbe qualsiasi ingresso addizionale della tastiera finchè non c'è il ritorno del responso dal computer centrale. Si noti che l'operatore saprà che l'ingresso è stato inviato poichè la spia BUSY non si accenderà. L'operatore saprà anche quando la tastiera è stata esclusa (il programma ignora gli ingressi della tastiera), poichè gli ingressi non compariranno sul display e la spia READY sarà spenta.

CORREZIONE DEGLI ERRORI DI TASTIERA

Gli ingressi non corretti sono un problema ovvio. Se l'operatore riconosce un errore può impiegare il tasto CLEAR per eseguire la correzione. L'operatore, comunque, commetterà un certo numero di errori senza riconoscerli. La maggior parte dei numeri di carta di credito comprendono un digit di autocontrollo; il terminale potrebbe controllare il numero prima di consentire di inviarlo al computer centrale. Questa fase risparmierebbe il computer centrale dallo spreco di elaborazione prezioso in virtù del controllo del numero.

Questo richiede, comunque, che il terminale abbia qualche modo di informare l'operatore dell'errore, forse accendendo un display oppure mediante qualche altro indicatore speciale che sicuramente l'operatore è in grado di notare. Alcuni terminali semplicemente si nota che la spia BUSY si è spenta senza che sia stata ricevuta una risposta. L'operatore si aspetta quindi di provare ancora l'ingresso.

Anche molti guasti dell'attrezzatura sono possibili. Accanto ai display, tastiera e processore esistono ora i problemi degli errori delle comunicazioni o guasti e guasti del computer centrale.

La trasmissione dei dati probabilmente dovrà comprendere qualche tipo di controllo di errore e procedure di correzione. Alcune possibilità sono:

CORREZIONE DEGLI ERRORI DI TRASMISSIONE

1) La parità fornisce una possibilità di rivelazione di errore ma non il meccanismo di correzione. Il ricevitore dovrà in qualche modo poter richiedere la ritrasmissione ed il mittente dovrà conservare una copia dei fatti fino a che non è avvenuta una trasmissione corretta.
   La parità, comunque, è molto semplice da realizzare.

2) Messaggi brevi possono impiegare schemi più elaborati. Per esempio la risposta si/no ad un terminale potrebbe essere codificata in modo da fornire una rivelazione di errore e la possibilità di correzione.

3) Un riconoscimento ed un limitato numero di recuperi potrebbero far scattare un indicatore che informerebbe l'operatore di un guasto delle comunicazioni (inabilità di trasferimento di un messaggio senza errori) oppure di un guasto al computer centrale (nessuna risposta a tutti i messaggi per un certo periodo di tempo). Tale schema, assieme al LAMP TEST, permetterebbe una diagnosi di guasto abbastanza semplice.

Un indicatore delle comunicazioni o di guasto del computer centrale potrebbe anche «aprire» il terminale, cioè permettergli di accettare un altro ingresso. Questo è necessario se il terminale non accetterà ingressi mentre è in corso una verifica. Il terminale può anche aprirsi dopo un certo tempo massimo di ritardo. Ingressi particolari potrebbero essere riservati come diagnostici, cioè certi numeri di carta di credito potrebbero essere impiegati per controllare il funzionamento interno del terminale ed il test dei display.

ANALISI DELLA DEFINIZIONE DEL PROBLEMA

La definizione del problema è una parte importante dello sviluppo del software come pure di qualunque altro problema ingegneristico. Si noti che essa non richiede nessuna programmazione o conoscenza del computer, piuttosto, essa si basa sulla comprensione del sistema e riflette la mentalità ingegneristica. I microprocessori possono offrire una flessibilità che il progettista può impiegare per fornire un range di caratteristiche precedentemente non disponibili.

La definizione del problema è indipendente da qualsiasi particolare computer, linguaggio del computer o sistema di sviluppo. Essa comunque, dovrebbe fornire le basi per stabilire il tipo o la velocità del computer che l'applicazione richiederà e quale tipo di compromesso hardware-software il progettista può impiegare. Lo stadio di definizione del problema è realmente indipendente dal fatto che il computer sia impiegato pienamente sebbene una conoscenza delle possibilità del computer possano aiutare il progettista suggerendogli una possibile realizzazione delle procedure.

PROGETTO DEL PROGRAMMA

Il progetto è uno stadio in cui la definizione del problema viene formulata sotto forma di programma. Se il programma è piccolo e semplice questo stadio può coinvolgere poco più della scrittura di una pagina di diagramma di flusso. Se il programma è grande o molto complesso il progettista può richiedere di considerare dei metodi più sofisticati.

Si discuterà la programmazione modulare mediante diagramma di flusso, la programmazione strutturata ed il progetto top-down. Si cercherà per quanto possibile di indicare la filosofia di questi metodi assieme ai loro svantaggi e vantaggi. Comunque non si difenderà nessun metodo in particolare, poichè non è chiaro che un metodo è sempre superiore a tutti gli altri. Si ricorderà che il traguardo è quello di produrre un buon sistema di lavoro e non di seguire religiosamente una metodologia od un'altra.

PRINCIPI BASE DEL PROGETTO DEL PROGRAMMA

Comunque tutte le metodologie hanno in comune gli stessi ovvi principi. Molti di questi ultimi sono gli stessi principi che si applicano a qualsiasi tipo di progetto, come:

1) Procedere in piccole fasi. Non cercare di fare troppo in una volta.
2) Dividere i grossi lavori in piccoli compiti separati dal punto di vista logico.
3) Mantenere il flusso di controllo più semplice possibile così da rendere più facile la ricerca degli errori.
4) Utilizzare illustrazioni o descrizioni a parole. Questo è il grande vantaggio dei diagrammi di flusso.
5) Accentuare prima di tutto la chiarezza e semplicità. Si può migliorare (se necessaria) una volta che il sistema è operante.
6) Procedere attraverso un modo sistematico. Impiegare liste di controllo oppure procedure standard.
7) Non tentare il fato. Non impiegare metodi di cui non si è sicuri oppure usarli con molta cautela. Fare attenzione alle situazioni che possono generare confusione e renderle chiare appena possibile.

8) Si ricordi che il sistema deve essere verificato, convalidato e mantenuto. Pianificare questi stadi successivi.
9) Impiegare tecnologia e metodi semplici e consistenti. La non ripetibilità non è un errore nel progetto del programma, la complessità non è una virtù.
10) Il progetto deve essere completamente formulato prima d'iniziare la codifica. Resistere alla tentazione di iniziare con la scrittura delle istruzioni; questo non avrebbe più senso di compilare una lista delle parti costituenti o della costruzione dello schema circuitale prima di conoscere esattamente come sarà il sistema.

DIAGRAMMI DI FLUSSO

La costruzione dei diagrammi di flusso (flowcharting) è il più noto di tutti i metodi di progetto del programma. I libri di testo della programmazione descrivono come i programmatori completano innanzi tutto la scrittura dei diagrammi di flusso e poi iniziano la scrittura effettiva del programma. Infatti pochi programmatori hanno lavorato in questo modo ed i diagrammi di flusso sono stati considerati uno scherzo oppure una seccatura per i programmatori piuttosto che un metodo di progetto. Si cercherà di descrivere i vantaggi e gli svantaggi dei diagrammi di flusso e di mostrare il ruolo di questa tecnica nel progetto del programma.

VANTAGGI DEI DIAGRAMMI DI FLUSSO

Il vantaggio base del diagramma di flusso è che esso fornisce una rappresentazione illustrata. Comunemente tale rappresentazione è molto più significativa di descrizioni scritte. Il progettista può visualizzare l'intero sistema e vedere i legami fra le varie parti. Errori ed incompatibilità spesso risaltano invece di risultare celate in una pagina stampata. Al meglio di diagramma di flusso è una immagine dell'intero sistema.

Alcuni dei vantaggi più specifici dei diagrammi di flusso sono:

1) Esistono dei simboli standard (Vedere Figura 13-7) cosicchè la forma dei diagramma di flusso è largamente riconosciuta.
2) Il diagramma di flusso può essere compreso da non esperti di programmazione.
3) Il diagramma di flusso può essere impiegato per dividere l'intero progetto in compiti più semplici. Il diagramma di flusso può poi essere esaminato per valutare lo stato di avanzamento globale.
4) Il diagramma di flusso mostra la sequenza di operazioni e perciò può aiutare nella localizzazione della sorgente di errori.
5) La tecnica dei diagrammi di flusso è ben nota in altre aree diverse dalla programmazione.

SVANTAGGI DEI DIAGRAMMI DI FLUSSO

Questi vantaggi sono tutti importanti. Non ci sono dubbi che la tecnica dei diagrammi di flusso continui ad essere largamente usata. Ci si soffermerà ad analizzare alcuni svantaggi della tecnica dei diagrammi di flusso come metodo di progetto del programma, per esempio:

1) I diagrammi di flusso sono difficoltosi da progettare, disegnare o cambiare tranne che nelle situazioni più semplici.
2) Non esiste un modo semplice per verificare o convalidare un diagramma di flusso.
3) I diagrammi di flusso tendono a divenire ingombranti. I progettisti trovano difficoltà a realizzare un bilanciamento tra l'importo di dettagli richiesti per rendere pratico un diagramma di flusso e l'importo che rende il diagramma di flusso un po' meglio della lista del programma.
4) I diagrammi di flusso mostrano solo l'organizzazione del programma.
Essi non mostrano l'organizzazione dei dati oppure la struttura dei moduli di ingresso/uscita.
5) I diagrammi di flusso non aiutano con hardware oppure non risolvono problemi di timing oppure suggeriscono come possono risolversi questi problemi.

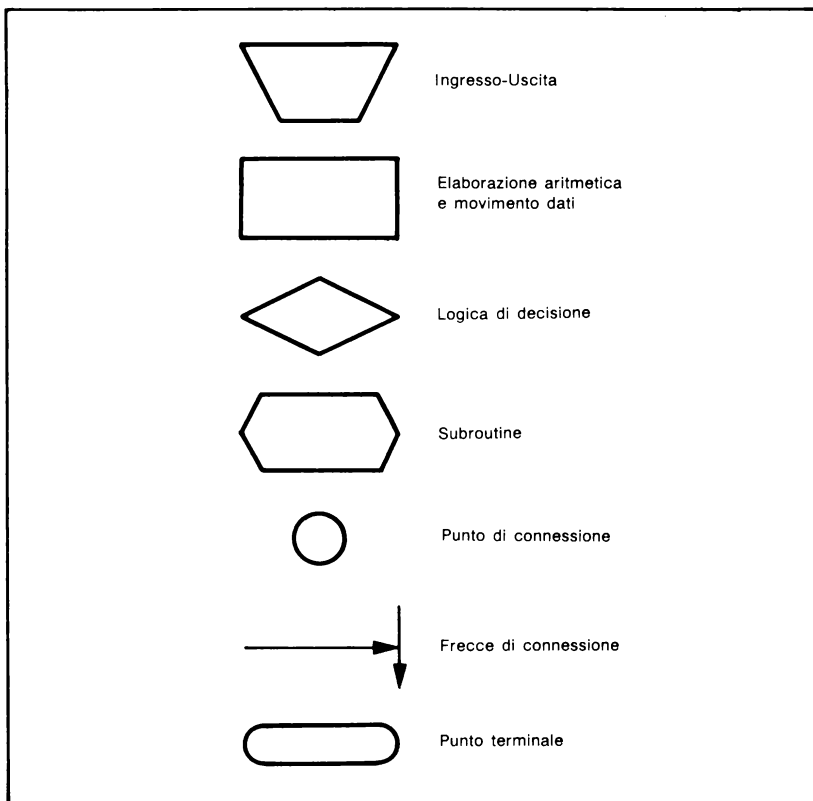



Figura 13-7
Simboli Del Diagramma Di Flusso Standard

Così la tecnica dei diagrammi di flusso è potente sì ma non troppo estendibile. I diagrammi di flusso sono comodi per la documentazione del programma poichè hanno una forma standard e sono comprensibili ai non programmatori. Come strumento di progetto, comunque, i diagrammi di flusso non possono fornire molto di più della linea di partenza; il programmatore non può verificare un diagramma di flusso dettagliato e spesso il diagramma di flusso è più difficoltoso da progettare dello stesso programma.

ESEMPI

Risposta ad un Interruttore

DIAGRAMMA DI FLUSSO DEL SISTEMA INTERRUTTORE DISPLAY

Questo semplice compito in cui un singolo interruttore accende una luce per un secondo è facile da rendere in diagramma di flusso. Infatti questi compiti sono tipici esempi dei libri di diagrammi di flusso sebbene essi formino una piccola parte della maggior parte dei sistemi. Qui la struttura dati è così semplice che può essere sicuramente ignorata.

La Figura 13-8 è il diagramma di flusso. C'è una piccola difficoltà nella decisione della quantità di dettagli richiesti. Il diagramma di flusso dà un'immagine veritiera della procedura che chiunque potrebbe comprendere.

si noti che i diagrammi di flusso più comodi possono ignorare le variabili del programma e formulare direttamente delle domande. Naturalmente spesso qui sono necessari dei compromessi. Talvolta sono convenienti due versioni del diagramma di flusso — una versione generale in linguaggio da profano, che sarà utile ai non programmatori, ed una versione per programmatori in termini di variabili del programma che sarà utile ad altri programmatori.

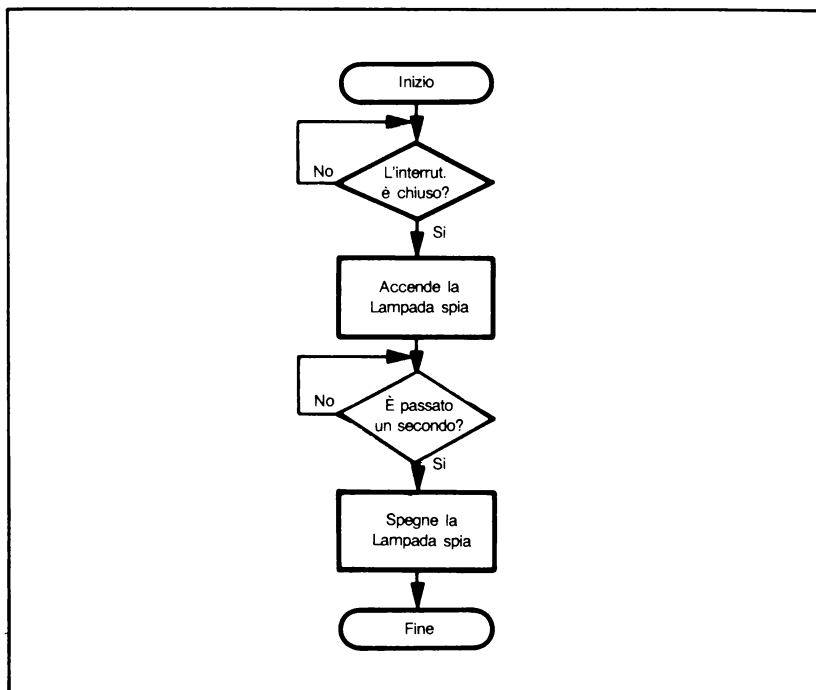



Figura 13-8
Diagramma Di Flusso Della Risposta Di Un Secondo Ad Un Interruttore

Il Caricatore di Memoria Basato sugli Interruttori

DIAGRAMMA DI FLUSSO DEL CARICATORE DI MEMORIA BASATO SULL'INTERRUTTORE

Questo sistema (ci si riferisca alla Figura 13-3) è considerevolmente più complesso dell'esempio precedente e coinvolge molte più decisioni. Il diagramma di flusso (vedere Figura 13-9) è molto difficoltoso da scrivere e non è chiaro come nell'esempio precedente. In questo esempio si elabora il problema poichè non c'è modo di verificare o convalidare il diagramma di flusso.

Il diagramma di flusso della Figura 13-9 coinvolge i miglioramenti suggeriti come parte della definizione del problema. Chiaramente questo diagramma di flusso comincia ad essere ingombrante e perde i suoi vantaggi su una descrizione della scrittura. Aggiungendo altre caratteristiche che definiscono il significato degli ingressi con le spie di stato e permettono all'operatore di controllare ingressi ulteriori dopo il completamento dell'ingresso corrente si potrebbe rendere il diagramma di flusso anche più complesso. La scrittura di diagrammi di flusso interi dall'inizio diventerebbe difficoltosa. Comunque quando un programma è stato scritto il diagramma di flusso è comodo come documentazione.

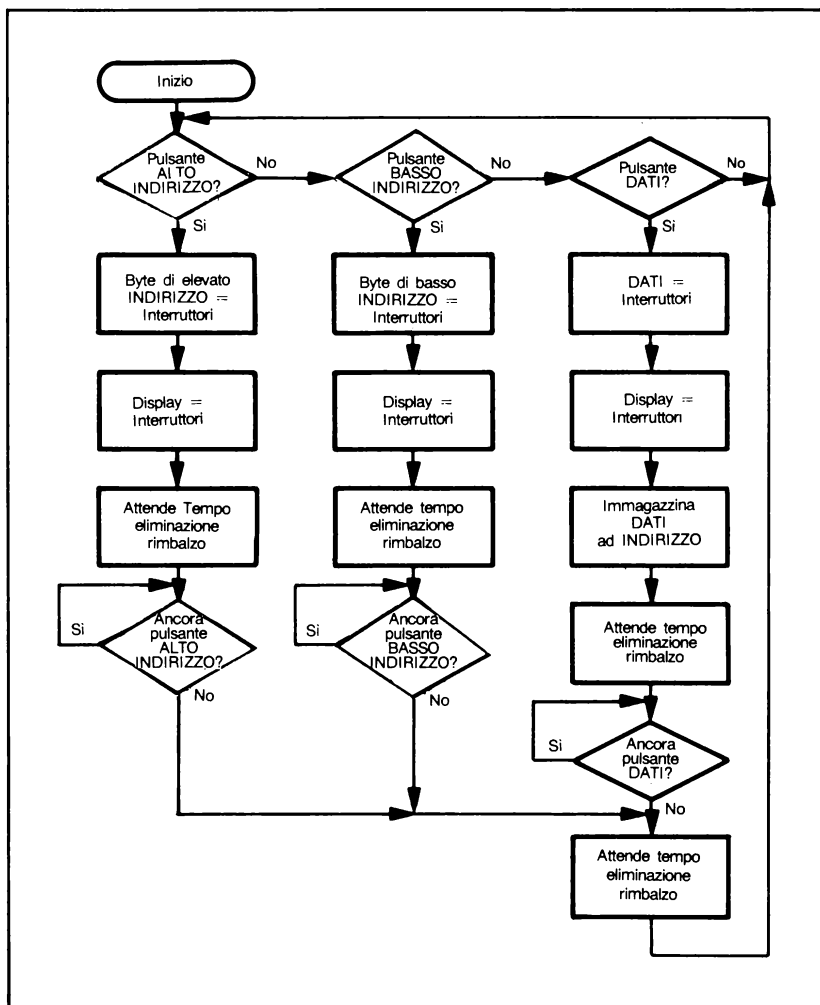



Figura 13-9
Diagramma Di Flusso Di Un Caricatore Di Memoria Basato Sull'Interruttore

Il Terminale di Verifica del Credito

DIAGRAMMA DI FLUSSO DELLA VERIFICA DEL CREDITO

In questa applicazione (vedere le Figure da 13-4 a 13-6) il diagramma di flusso sarà anche più complesso del caso del caricatore di memoria basato sull'interruttore. Qui l'idea migliore è quella di costruire il diagramma di flusso delle varie sezioni separatamente cosicchè il diagramma di flusso rimane maneggevole. Comunque la presenza di strutture dati (come nel display multi-digit ed i messaggi) determinerà del vuoto tra diagramma di flusso e programma.

SEZIONI DEL DIAGRAMMA DI FLUSSO

Verranno considerate ora alcune delle sezioni. La Figura 13-10 mostra il processo d'ingresso da tastiera per i tasti dei digit. Il programma deve andare a prendere i dati dopo ogni strobe e posizionare il digit nell'array del display se c'è un posto disponibile. Se ci sono già dieci digit nell'array il programma ignora semplicemente l'ingresso.

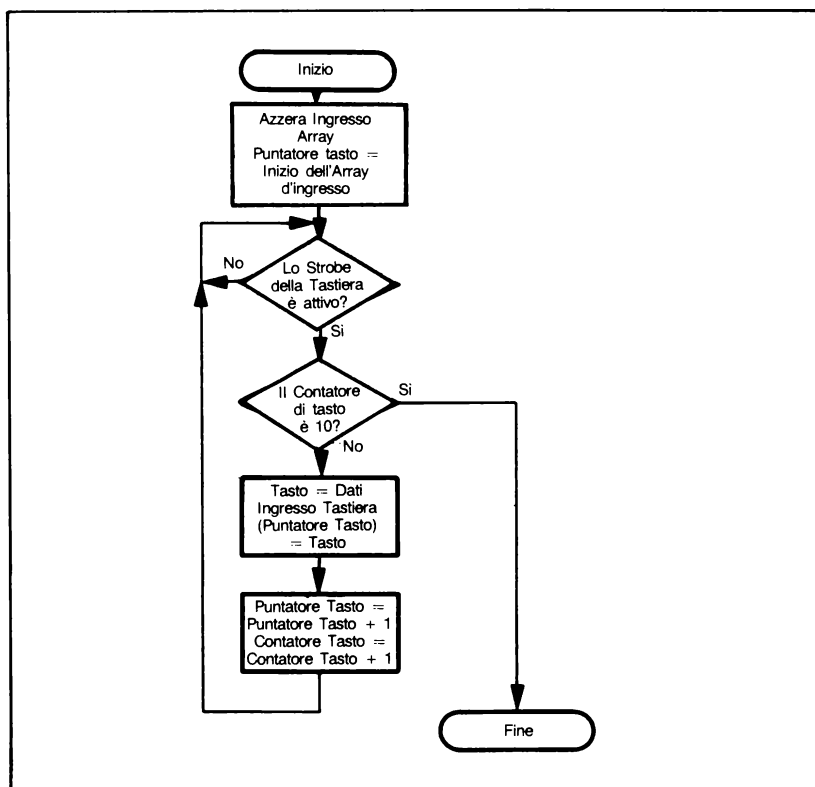



Figura 13-10
Diagramma Di Flusso Del Processo D'Ingresso Da Tastiera

Il programma effettivo dovrà maneggiare i display contemporaneamente. Si noti che sia il software che l'hardware devono disattivare lo strobe della tastiera dopo che il processore legge un digit.

La Figura 13-11 aggiunge il tasto SEND. Questo tasto, naturalmente è a scelta. Il terminale potrebbe inviare i dati non appena l'operatore fa entrare un numero completo. Comunque questa procedura non fornirà all'operatore un mezzo per controllare l'intero programma. Il diagramma di flusso con il tasto SEND è più complesso perchè ci sono due alternative:

1) Se l'operatore non ha fatto entrare dieci digit il programma ignorerà il tasto SEND e posizionerà qualunque altra chiave all'ingresso.
2) Se l'operatore ha fatto entrare dieci digit il programma deve rispondere alla chiave SEND mediante il trasferimento del controllo alla routine SEND ed ignora tutti gli altri tasti.

Si noti che il diagramma di flusso è diventato molto più difficoltoso da organizzare e da eseguire. Ci sono anche modi non ovvi per controllare il diagramma di flusso.

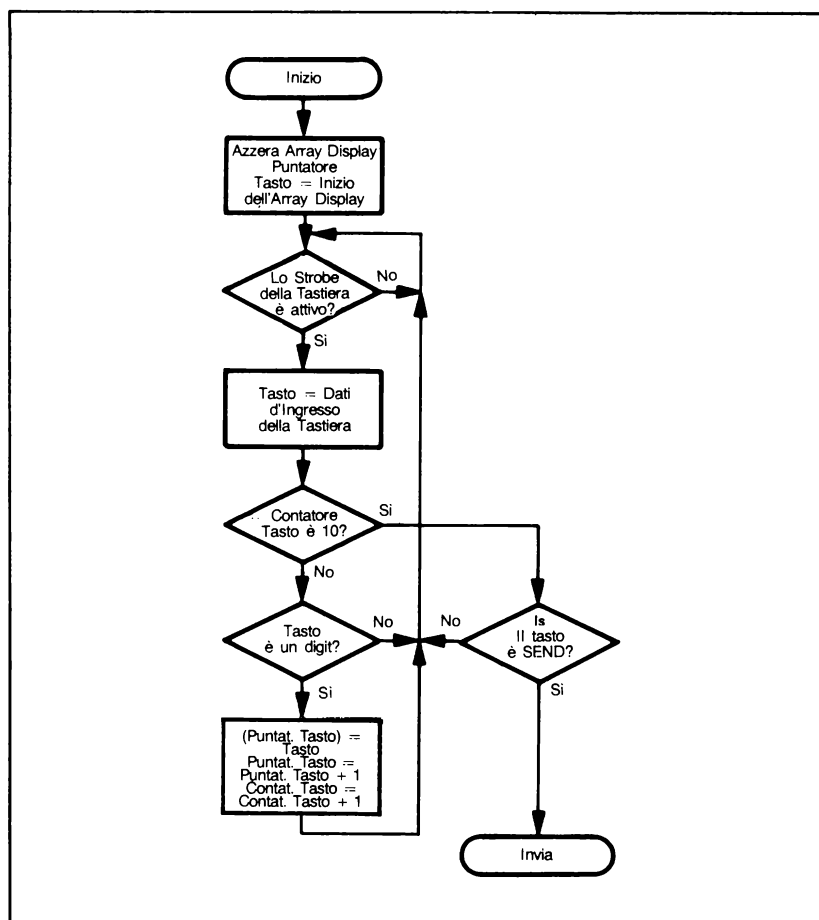



Figura 13-11
Diagramma Di Flusso Del Processo D'Ingresso Da Tastiera Con Il Tasto SEND

La Figura 13-12 mostra il diagramma di flusso del processo di ingresso da tastiera con tutti i tasti funzionali. In questo esempio il flusso di controllo non è realizzato con mezzi semplici. Chiaramente è necessaria qualche descrizione scritta. L'organizzazione dello schema di diagrammi di flusso complessi richiede un'attenta pianificazione. È stato seguito il processo per aggiungere caratteristiche al diagramma di flusso una alla volta, ma questo si risolverà in una grande quantità di ripetizioni grafiche. Ancora ci si ricorderà che, attraverso il processo di ingresso della tastiera, il programma deve anche rinfrescare i display se questi sono multiplexed e non controllati da registri di scorrimento oppure da altro hardware.

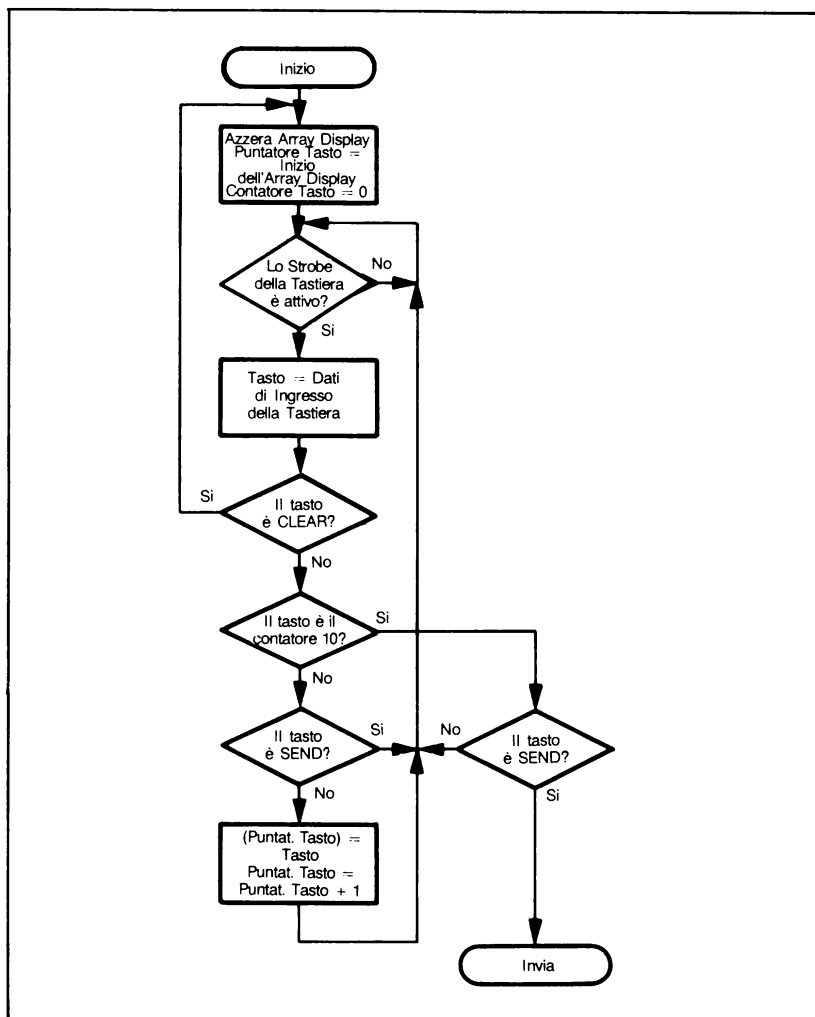



Figura 13-12
Diagramma Di Flusso Del Processo D'Ingresso Da Tastiera Con I Tasti Di Funzione

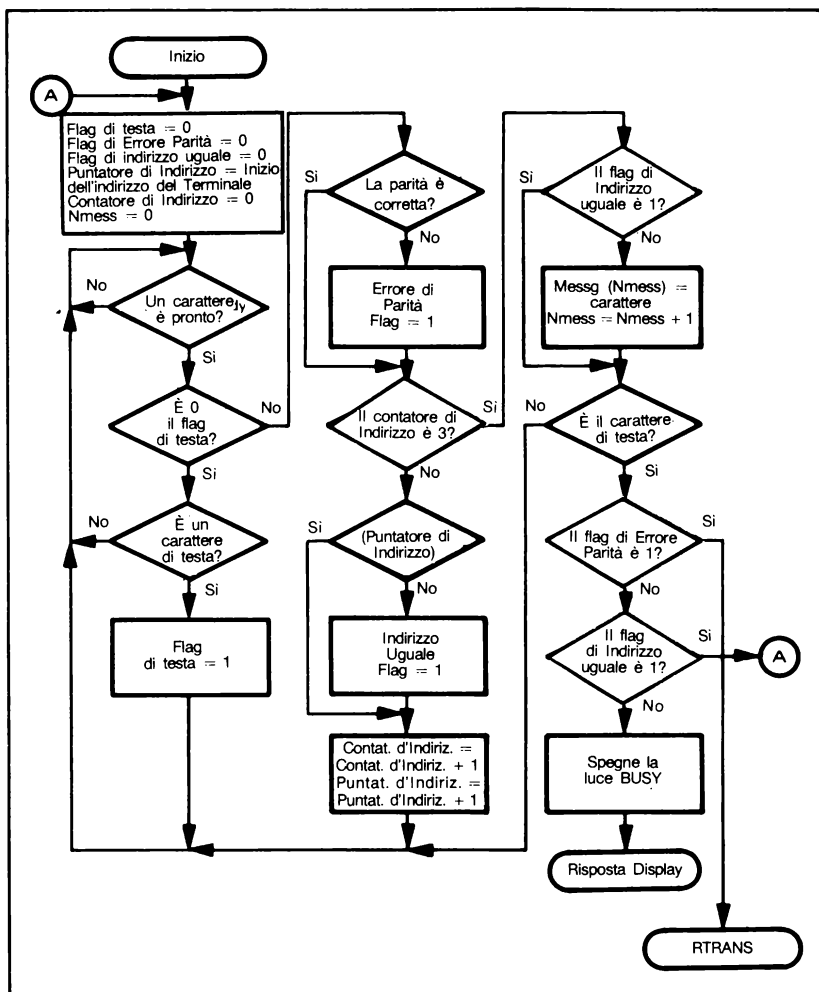



Figura 13-13
Diagramma Di Flusso Della Routine Di Ricezione

La Figura 13-13 è il diagramma di flusso della routine di ricezione. Si assume che la conversione seriale/parallelo ed il controllo di errore è fatto in hardware (cioè mediante un'UART). Il processore deve:

1) Osservare la testa (si assume che sia un singolo carattere).
2) Leggere l'indirizzo di destinazione (si assume lungo tre caratteri) e vede se il messaggio ha significato per questo terminale, cioè se i tre caratteri sono in accordo con l'indirizzo del terminale.
3) Attendere il carattere di coda.
4) Se il messaggio ha significato per il terminale spegne la spia BUSY e va alla routine DISPLAY ANSWER.
5) Nell'eventualità di qualunque errore richiede la ritrasmissione andando alla routine RTRAN.

La routine coinvolge un grande numero di decisioni ed il diagramma di flusso non è semplice nè ovvio.

Chiaramente ci si è molto allontanati dal semplice diagramma di flusso (Figura 13-8) del primo esempio. Un set completo di diagrammi di flusso per il terminale di transizione sarebbe il compito maggiore. Esso considererebbe di alcuni diagrammi correlati con logica complessa e richiederebbe un grande sforzo. Tale sforzo sarebbe difficoltoso come la scrittura preliminare di un programma ma così non comodo poichè non si può controllare su computer.

PROGRAMMI MODULARI

Una volta che i programmi diventano grandi e complessi i diagrammi di flusso sono uno strumento di progetto non soddisfacente. Comunque la definizione del problema ed il diagramma di flusso possono originare un'idea su come dividere il programma in ragionevoli compiti secondari. La divisione dell'intero programma in compiti secondari o moduli è detta «programmazione modulare». Chiaramente la maggior parte dei programmi presentati ai capitoli precedenti, sarebbero dei moduli tipici in un grosso programma di sistema. I problemi che il progettista incontra nella programmazione modulare sono dovuti a come dividere il programma in moduli e come mettere insieme i moduli.

VANTAGGI DELLA PROGRAMMAZIONE MODULARE

I vantaggi della programmazione modulare sono ovvi:

1) Un modulo singolo è facile da scrivere, verificare e convalidare come un intero programma.
2) Probabilmente un modulo è impiegato in molte posizioni ed in altri programmi, particolarmente se è generale in modo ragionevole ed esegue un compito comune. È possibile costruire una biblioteca dei moduli standard.
3) La programmazione modulare permette al programmatore di dividere i compiti ed impiegare i programmi scritti precedentemente.
4) Le variazioni possono essere incorporate in un modulo piuttosto che in un intero sistema.
5) Gli errori possono essere isolati e quindi attribuiti ad un singolo modulo.
6) La programmazione modulare dà un'idea di quanto progresso è stato fatto e di quanto lavoro resta.

SVANTAGGI DELLA PROGRAMMAZIONE MODULARE

L'idea della programmazione modulare è talmente ovvia che i suoi svantaggi sono spesso ignorati. Questi comprendono:

1) L'adattamento assieme dei moduli possono essere il problema più grave, particolarmente se persone diverse scrivono i moduli.
2) I moduli richiedono una documentazione molto accurata, poichè essi possono influenzare le varie parti del programma come per esempio le strutture dati impiegate da tutti i moduli.
3) La convalida e la verifica dei moduli è difficile, separatamente poichè gli altri metodi possono produrre i dati utilizzati dal modulo che è stato verificato ed anche altri moduli possono impiegare i risultati.
4) I programmi possono essere difficoltosi da modularizzare in qualsiasi modo ragionevole. Se si modularizza il programma in modo insufficiente, l'integrazione sarà molto difficile, poichè quasi tutti gli errori ed i cambiamenti risultanti coinvolgeranno diversi blocchi.
5) I programmi modulari spesso richiedono ulteriore tempo e memoria, poichè dei blocchi possono ripetere le funzioni.

Perciò, mentre la programmazione modulare è certamente un miglioramento sul semplice tentativo di scrittura di un intero programma dall'inizio, essa peraltro ha alcuni svantaggi.

ESEMPI

Risposta ad un Interruttore

Questo semplice programma può essere diviso in due moduli:

MODULARIZZAZIONE DEL SISTEMA INTERRUTTORE DISPLAY

Il Modulo 1 attende che l'interruttore sia chiuso ed in risposta accende il display.

Il Modulo 2 fornisce un ritardo di un secondo.

Il Modulo 1 è probabile che sia specifico del sistema, poichè esso dipende da come l'interruttore ed il display sono connessi. Il Modulo 2 sarà di uso generale poichè molti compiti richiedono ritardi. Chiaramente sarà vantaggioso avere un modulo di ritardo standard che fornisca ritardi di lunghezza variabile. Il modulo richiederà una documentazione accurata cosicchè si conoscerà come specificare la lunghezza del ritardo, come chiamare il modulo e quali registri e locazioni di memoria sono influenzate dal modulo stesso.

Una versione generale del Modulo 1 sarebbe molto meno pratica poichè dovrebbe essere compatibile con diversi tipi e connessioni di interruttori e display o spie luminose.

Si troverebbe probabilmente più semplice scrivere un modulo per una particolare configurazione di interruttori e spie luminose piuttosto che provare ad impiegare una routine standard. Si noti la differenza tra questa situazione ed il Modulo 2.

Il Caricatore di Memoria Basato sugli Interruttori

Il caricatore di memoria basto sull'interruttore è difficile da rendere modulare poichè tutti i compiti della programmazione dipendono dalla configurazione hardware ed i compiti sono così semplici che non vale la pena definirli. Il diagramma di flusso della Figura 13-9 suggerisce che un modulo può essere quello che attende che l'operatore prema uno dei tre pulsanti.

MODULARIZZAZIONE DEL CARICATORE DI MEMORIA BASATO SULL'INTERRUTTORE

Alcuni altri moduli possono essere:

1) Un modulo di ritardo che fornisce il ritardo richiesto per eliminare il rimbalzo degli interruttori.
2) Un modulo interruttore - display che legge i dati dagli interruttori e li invia ai display.
3) Un modulo di LAMP TEST.

Tali moduli altamente dipendenti dal sistema probabilmente non sono utili comunemente. Questo è un esempio in cui la programmazione modulare non offre grandi vantaggi.

Il Terminale di Verifica

Il terminale di verifica, d'altra parte, si presta molto bene alla programmazione modulare. L'intero sistema può essere facilmente diviso in .tre moduli principali:

MODULARIZZAZIONE DEL TERMINALE DI VERIFICA

1) Modulo Tastiere e Display.
2) Modulo di trasmissione dati.
3) Modulo di ricezione dati.

Un modulo tastiera e display generale potrebbe manipolare molti sistemi basati su tastiera e display. I sotto - moduli che eseguono tali compiti sono:

1) Riconoscimento dell'ingresso di un nuovo tasto ed accettazione dei dati.

2) Azzeramento dell'array in risposta al tasto CLEAR.
3) Ingresso dei digit nella memoria.
4) Riconoscimento del tasto terminale o SEND.
5) Visualizzazione dei digit (displaying).

Benchè le interpretazioni dei tasti ed il numero dei digit varieranno, i processi di ingresso di base, immagazzinamento dati ed esposizione dati saranno gli stessi per molti programmi. Alcuni tasti di funzione devono considerare quali moduli saranno convenienti in altre applicazioni e porre un'accurata attenzione a quei moduli.

Il modulo di trasmissione dati potrebbe anche essere diviso in sub-moduli come:

1) Aggiunta del carattere di testa.
2) Caratteri da trasmettere che la linea di uscita può manipolare.
3) Generazione tempi di ritardo tra bit o caratteri.
4) Aggiunta del carattere di coda.
5) Controllo per guasti di trasmissione, cioè non riconoscimento ovvero inabilità a trasmettere senza errori.

Il modulo di ricezione dati può comprendere i seguenti sub-moduli:

1) Osservazione del carattere di testa.
2) Controllo dell'indirizzo di destinazione del messaggio rispetto all'indirizzo del terminale.
3) Immagazzinamento ed interpretazione del messaggio.
4) Osservazione del carattere di coda.
5) Generazione dei ritardi di bit o di carattere.

ANALISI DELLA PROGRAMMAZIONE MODULARE

La programmazione modulare può essere molto potente se si seguono le seguenti regole:

REGOLE PER LA PROGRAMMAZIONE MODULARE

1) Impiegare moduli da 20 a 50 righe. Moduli più corti sono normalmente uno spreco di tempo, mentre moduli più lunghi raramente sono generali e possono essere difficoltosi da integrare.
2) Si cerchi di rendere i moduli ragionevolmente generali. Si differenzi tra le caratteristiche comuni come il codice ASCII oppure i formati di trasmissione asincrona che saranno gli stessi per molte applicazioni e le identificazioni di tasti, il numero di display oppure il numero di caratteri di un messaggio che probabilmente saranno unici per una particolare applicazione. Operare il cambiamento degli ultimi parametri semplici.
   Maggiori scambi come codici di carattere diversi sarebbero manipolati da moduli separati.
3) Si impieghi ulteriore tempo su moduli come display manipolatori di display, manipolatori di tastiere, ecc. che saranno comodi in altri progetti od in molte situazioni diverse del programma attuale.
4) Si cerchi di mantenere i moduli più distinti e logicamente separati possibile.
5) Non si cerchi di modularizzare i compiti semplici dove la riscrittura dell'intero compito può essere più facile dell'assemblaggio o modifica del modulo.

PROGRAMMAZIONE STRUTTURATA

Con quale criterio mantenere i moduli distinti ed arrestarli dall'interazione di uno con l'altro? Come scrivere un programma che ha una sequenza di cancellazione di operazioni in modo da selezionare e correggere gli errori? Una risposta è di impiegare il metodo noto come «programmazione strutturata» dove ogni parte del programma consiste di elementi da un set limitato di strutture ed ogni struttura ha un singolo ingresso ed una singola uscita.

La Figura 13-14 mostra un diagramma di flusso di un programma non strutturato. Se si verifica un errore nel Modulo B ci sono cinque possibili sorgenti di questo errore. Non solo si deve controllare ogni sequenza possibile ma occorre anche assicurarsi che qualunque cambiamento fatto per correggere l'errore non influenzi qualunque altra sequenza. Il risultato comune è che la verifica è spesso disastrosa. Ogni volta che si pensa che la situazione sia sotto controllo si trova un ulteriore errore.

STRUTTURE BASE DELLA PROGRAMMAZIONE STRUTTURATA

La risposta a questo problema è di stabilire una sequenza di operazioni di azzeramento in modo che si possano isolare errori. Tale sequenza di azzeramento impiega moduli ad ingresso singolo ed uscita singola. I moduli base che sono richiesti sono:

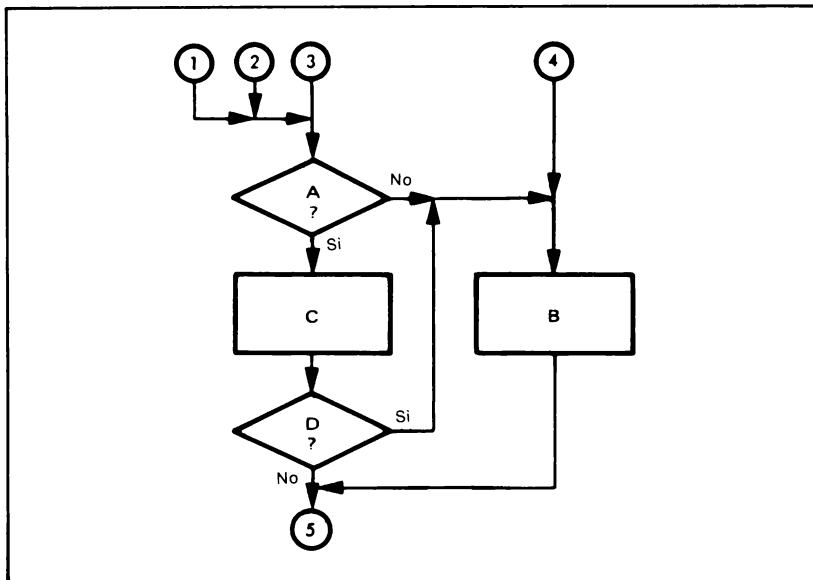



Figura 13-14
Diagramma Di Flusso Di Un Programma Non Strutturato

1) Una sequenza ordinaria cioè una struttura lineare in cui gli statement o le strutture sono eseguite consecutivamente. Nella sequenza:
   PI
   P2
   P3
   il computer esegue P1 per prima, P2 per seconda, P3 per terza. P1, P2, P3 possono essere istruzioni singole di un intero programma.
2) Una struttura condizionale.
   Una struttura comune è «Se A allora P1 altrimenti P2» («if A then P1 else P2») dove A è una condizione e P1, P2 sono programmi. Il computer esegue P1 se A è vera e P2 se è falsa. La Figura 13-15 mostra, la logica di questa struttura. Si noti che la struttura ha un singolo ingresso ed una singola uscita. Non c'è altro modo per entrare oppure per far uscire P1 o P2 se non attraverso la struttura.
3) Una struttura ciclica.
   La comune struttura a ciclo è «esegue P finchè A» («do P while A») dove A è una condizione e P è un programma. Il computer controlla A ed esegue P se A è vera. Questa struttura

(Vedere Figura 13-16) ha pure un singolo ingresso ed una singola uscita. Si noti che il computer non eseguirà P se A è originariamente falsa poichè il valore di A è controllato prima dell'esecuzione di P.

Si notino le seguenti caratteristiche della programmazione strutturata:

1) Sono consentite solo tre strutture di base.
2) Le strutture possono essere modificate a qualunque livello di complessità cosicchè qualunque programma può, a turno, contenere qualsiasi struttura.
3) Ogni struttura ha un singolo ingresso ed una singola uscita.

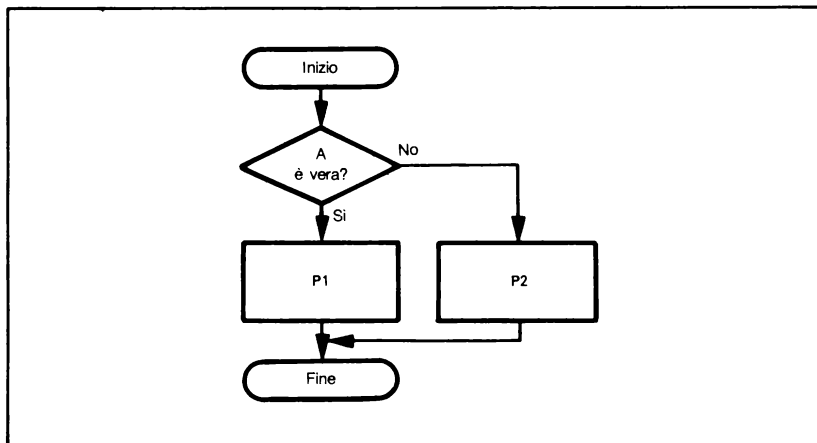



Figura 13-15
Diagramma Di Flusso Di Una Struttura Se - Allora - Altrimenti

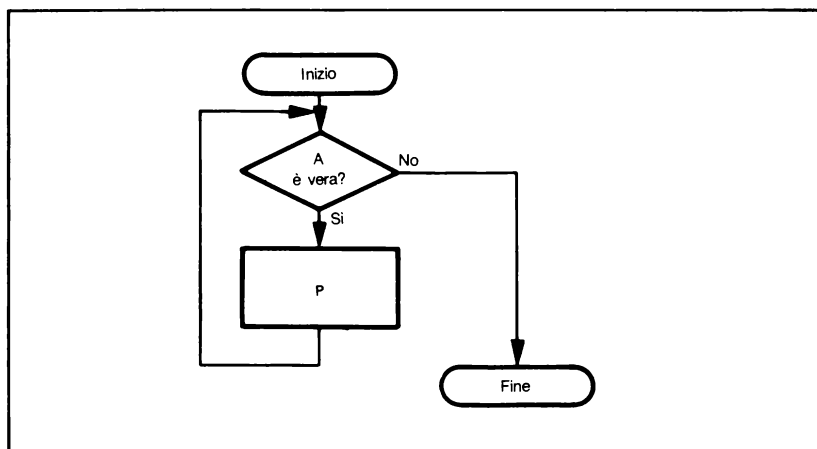



Figura 13-16
Diagramma Di Flusso Della Struttura Esegue - Finchè

Alcuni esempi della struttura condizionale illustrati in Figura 13-15 sono:

1) P2 comprende:
   se X $\geq$ 0 allora NPOS = NPOS + 1
   altrimenti NNEG = NNEG + 1

P1 e P2 sono entrambi singoli statement.

2) P2 omesso:
   se X $\neq$ 0 allora y = 1/X

Qui non viene condotta nessuna operazione se A (X$\neq$ 0) è falsa. P2 ed «altrimenti» possono essere omesse in questo caso.

Alcuni esempi della struttura ciclo illustrata in Fig. 13-16 sono:

1) Esegue la somma di interi da 1 ad N.

```
I = 0
SUM = 0
esegue finchè I < N
        I = I + 1
        SUM = SUM + 1
        fine
```

Il computer esegue il ciclo finchè I < N. Se N = 0 il programma dentro il «esegue finchè» non viene mai eseguito.

2) Conteggio dei caratteri di un array SENTENCE finchè si trova un periodo ASCII.

```
NCHAR = 0
esegue finchè SENTENCE (HCHAR) ≠ PERIOD
     NCHAR = NCHAR + 1
    fine
```

Il computer esegue il ciclo finchè il carattere in SENTENCE non è un periodo ASCII. Il conteggio è zero se il primo carattere è un periodo.

VANTAGGI DELLA PROGRAMMAZIONE STRUTTURATA

I vantaggi della programmazione strutturata sono:

1) La sequenza di operazioni è semplice da tracciare. Questo permette una facile verifica e convalida.
2) Il numero di strutture è limitato e la terminologia è standardizzata.
3) Le strutture possono essere facilmente realizzate in moduli.
4) Dal punto di vista teorico è stato dimostrato che un dato set di strutture è completo, cioè tutti i programmi possono essere scritti in termini di queste tre strutture.
5) La versione strutturata del programma è parzialmente audo-comentante ed abbastanza facile da leggere.
6) I programmi strutturati sono facili da descrivere con diagrammi di flusso od altri metodi grafici.
7) In pratica è stato dimostrato che la programmazione strutturata aumenta la produttività.

Fondamentalmente la programmazione strutturata forza molta più disciplina sul programmatore, rispetto a quella non strutturata. Il risultato è un'organizzazione dei programmi migliore o più sistematica.

SVANTAGGI DELLA PROGRAMMAZIONE STRUTTURATA

Gli svantaggi della programmazione strutturata sono:

1) Solo alcuni linguaggi ad alto livello (per esempio PL/M, PASCAL) accetteranno direttamente le strutture. Il programmatore deve perciò passare attraverso uno stadio di ulteriore trasformazione per convertire le strutture in un codice in linguaggio assembly. La versione strutturata del programma comunque è spesso comoda come documentazione.
moda come documentazione.
2) I programmi strutturati spesso hanno una esecuzione più lenta ed usano più memoria dei programmi non strutturati.
3) Limitando le strutture alle tre forme di base alcuni compiti risultano molto scomodi da eseguire. La completezza delle strutture significa soltanto che tutti i programmi possono essere realizzati con queste; questo non significa che tutti i programmi possano essere realizzati in modo efficiente e conveniente.
4) Le strutture standard sono spesso tali da creare confusione, per esempio le strutture annidate «se-allora-altrimenti» possono essere molto difficoltose da leggere poichè possono non fornire una indicazione chiara di dove una termina. Una serie di cicli «esegue-finchè» può anche essere difficile da leggere.
5) I programmi strutturati considerano solo la sequenza di operazioni di programma e non il flusso di dati. Perciò le strutture possono manipolare dati in modo maldestro.
6) Pochi programmatori sono abituati alla programmazione strutturata. Molti trovano le strutture standard maldestre e restrittive.

Qui non si vuole né difendere né scoraggiare l'impiego della programmazione strutturata. Essa costituisce un metodo sistematico di progetto di programma. In generale essa è molto comoda nelle situazioni seguenti.

QUANDO IMPIEGARE LA PROGRAMMAZIONE STRUTTURATA

1) Programmi molto lunghi, magari maggiori di 1000 istruzioni.
2) Applicazioni dove l'impiego della memoria non è critico.
3) Applicazioni di piccolo volume dove i costi dello sviluppo software, particolarmente verifica e convalida, sono fattori importanti.
4) Applicazioni che non coinvolgono strutture dati complesse.
5) Applicazioni che coinvolgono manipolazioni di stringhe, controllo di processo ed altri algoritmi piuttosto che semplici manipolazioni di bit.
6) Applicazioni dove si sta impiegando un linguaggio ad alto livello.

In futuro è verosimile aspettarsi che il costo delle memorie diminuirà, la dimensione media dei programmi del microprocessore tenderà ad aumentare ed il costo dello sviluppo software a diminuire. Perciò i metodi della programmazione strutturata, che diminuiscono i costi dello sviluppo software per programmi molto lunghi ma impiegano più memoria, ci si può aspettare che in futuro si accentuerà.

ESEMPI

Risposta ad un Interruttore

PROGRAMMAZIONE STRUTTURATA DEL SISTEMA INTERRUTTORE DISPLAY

La versione strutturata di questo esempio è:

```
INTERRUTTORE = APERTO
esegui finchè INTERRUTTORE = APERTO
  LEGGE INTERRUTTORE
  fine
DISPLAY = ACCESO
RITARDO 1
DISPLAY = SPENTO
```

Si assume che RITARDO è il modulo che fornisce un'attesa data dal suo parametro espresso in secondi.

Uno statement in un programma strutturato può effettivamente essere una subroutine. Comunque allo scopo di adeguarsi alle regole della programmazione strutturata, la subroutine non può avere nessun altra uscita all'infuori di quella che controlla il ritorno al programma principale.

Poichè «esegue finchè» controlla la condizione prima dell'esecuzione del ciclo si pone la variabile SWITCH ad OFF prima della partenza. La programmazione strutturata è chiara, leggibile e facile da verificare manualmente. Comunque essa probabilmente richiederebbe talvolta più memoria di quella di un programma non strutturato che non dovrebbe inizializzare SWITCH e potrebbe abbinare le procedure di lettura e verifica.

Il Caricatore di Memoria Basato sugli Interruttori

PROGRAMMAZIONE STRUTTURATA PER IL CARICATORE DI MEMORIA BASATO SULL'INTERRUTTORE

Il caricatore di memoria basato sugli interruttori è un problema più complesso di programmazione strutturata. Si può realizzare il diagramma di flusso di Figura 13-9 come segue (un * indica un commento):

```
*
* INIZIALIZZA LE VARIABILI
*
INDIRIZZO ALTO = 0
INDIRIZZO BASSO = 0
DATI = 0
*
* QUESTO PROGRAMMA HA UN ESEGUI FINCHÈ È SODDISFATTA UNA CONDIZIONE
* CHE È SEMPRE VERA, PERCIÒ IL SISTEMA CONTINUAMENTE ESEGUE
* IL PROGRAMMA CONTENUTO IN QUESTO CICLO DI ESEGUE FINCHÈ
*
esegui finchè 1 = 1
*
* PROVA PER IL PULSANTE INDIRIZZO ALTO, ESEGUE L'ELABORAZIONE RICHIESTA
* SE È CHIUSO
*
  se il PULSANTE INDIRIZZO ALTO = 1 allora
     INDIRIZZO ALTO = INTERRUTTORI
     DISPLAY = INTERRUTTORI
     RITARDO (TEMPO DI ELIMINAZIONE RIMBALZO)
     esegui finchè il PULSANTE INDIRIZZO ALTO = 1
       RITARDO (TEMPO DI ELIMINAZIONE RIMBALZO)
       fine
*
* PROVA PER IL PULSANTE INDIRIZZO BASSO; ESEGUE L'ELABORAZIONE INDIRIZZO
  BASSO SE
* ESSO È CHIUSO
*
  se il PULSANTE INDIRIZZO BASSO = 1 allora
     INDIRIZZO BASSO = INTERRUTTORI
     DISPLAY = INTERRUTTORI
     RITARDO (TEMPO DI ELIMINAZIONE RIMBALZO)
       esegui finchè il PULSANTE DI INDIRIZZO BASSO = 1
       RITARDO (TEMPO DI ELIMINAZIONE RIMBALZO)
       fine
```

```
*
* PROVA PER IL PULSANTE DATI ED IMMAGAZZINA I DATI NELLA MEMORIA
* SE ESSO È CHIUSO
*
  se il PULSANTE DATI = 1 allora
     DATI = INTERRUTTORI
     DISPLAY = INTERRUTTORI
     (INDIRIZZO ALTO INDIRIZZO BASSO) = DATI
     RITARDO (TEMPO DI ELIMINAZIONE RIMBALZO)
     esegui finchè il PULSANTE DATI = 1
       RITARDO (TEMPO DI ELIMINAZIONE RIMBALZO)
       fine
  fine
*
* IL FINE PRECEDENTE TERMINA L'
*    esegui finchè 1 = 1 CICLO
*
```

I programmi strutturati non sono facili da scrivere ma possono fornire una grande intuizione su tutta la logica del programma. Si può controllare la logica del programma strutturato per mezzo della scrittura precedente di qualunque codice effettivo.

Il Terminale di Verifica del Credito

PROGRAMMA STRUTTURATO PER IL TERMINALE DI VERIFICA DEL CREDITO

ROUTINE DI TASTIERA STRUTTURATA

Si osserverà l'ingresso della tastiera per il terminale di transazione. Si assumerà che l'array del display sia ENTRY, lo strobe di chiave della tastiera sia KEYSTROBE, ed i dati dei tasti della tastiera siano KEYIN. Il programma strutturato senza i tasti di funzione è:

```
NKEYS = 10
*
* AZZERA ENTRY ALL'INIZIO
*
  esegui finchè NKEYS > 0
  NKEYS = NKEYS - 1
  ENTRY (NKEYS) = 0
  fine
*
* ACQUISISCE UN INGRESSO COMPLETO DALLA TASTIERA
*
  esegui finchè NKEYS < 10
    se KEYSTROBE = ATTIVO allora
       KEYSTROBE = INATTIVO
       ENTRY (NKEYS) = KEYIN
       NKEYS = NKEYS + 1
    fine
```

Aggiungendo il tasto SEND significa che il programma deve ignorare ulteriori digit dopo che è stato completato l'ingresso e deve ignorare il tasto SEND finchè l'ingresso è completo. Il programma strutturato è:

```
NKEYS = 10
*
* AZZERA ENTRY ALL'INIZIO
*
  esegui finchè NKEYS 0
     NKEYS = NKEYS - 1
     ENTRY (NKEYS) = 0
     fine
```

```
*
* ATTENDE INGRESSO COMPLETO ED INVIA TASTO
*
  esegui finchè≠ SEND oppure NKEYS≠ 10
    se KEYSTROBE = ATTIVO allora
       KEYSTROBE = INATTIVO
       TASTO = KEYIN
*
* ACCETTA DIGIT SE NON È PRESENTE UN INGRESSO COMPLETO
*
       se NKEY≠10 E TASTO≠ SEND
          ENRY (NKEYS) = TASTO
          NKEYS = NKEYS + 1
    fine
```

L'aggiunta del tasto CLEAR permette al programma di azzerare originalmente l'ingresso mediante di un mutamento cioè ponendo NKEYS a 10 ed azzerando il tasto prima della partenza. Il programma strutturato deve anche cancellare i digit che sono stati precedentemente caricati. Il nuovo programma strutturato è:

```
*
* SIMULAZIONE DI AZZERAMENTO COMPLETO
*
NKEYS = 10
TASTO = CLEAR
*
* ATTENDE UN INGRESSO COMPLETO ED IL TASTO SEND
*
esegui finchè TASTO≠ SEND OPPURE NKEYS≠ 10
*
* AZZERA TUTTO L'INGRESSO SE IL TASTO CLEAR È PREMUTO
*
  se TASTO = CLEAR allora
     TASTO = 0
     esegui finchè NKEYS > 0
       NKEYS = NKEYS -1
       ENTRY (NKEYS) = 0
       fine
*
* ACCETTA DIGIT SE L'INGRESSO È INCOMPLETO
*
  se KEYSTROBE = ATTIVO allora
     KEYSTROBE = INATTIVO
     TASTO = KEYIN
  se TASTO < 10 ed NKEYS≠ 10 allora
     ENTRY (NKEYS) = TASTO
     NKEYS = NKEYS + 1
  fine
```

Si noti che il programma azzera il tasto CLEAR cosicchè esegue il processo di annullamento una volta sola.

Analogamente si può costruire un programma strutturato per una routine di ricezione. Un programma iniziale potrebbe osservare i caratteri di testa e coda. Si assumerà che RSTB è l'indicatore che un carattere è pronto. Il programma strutturato è:

```
*
*AZZERA IL FLAG DI TESTA ALL'INIZIO
*
HFLAG = 0
*
*ATTENDE I CARATTERI DI TESTA E CODA
*
esegui finchè è HFLAG = 0 oppure CHAR≠ TESTA
*
*ACCETTA CARATTERE SE PRONTO, OSSERVA IL CARATTERE DI TESTA
*
  se RSTB = ATTIVO allora
     RSTB = INATTIVO
     CHAR = INGRESSO
  se CHAR = TESTA allora
     HFLAG = 1
  fine
```

Ora si può aggiungere la sezione che controlla l'indirizzo del messaggio rispetto ai tre digit in TERMINAL ADDRESS (TERMADD). Se nessuno dei digit corrispondenti è uguale viene posto ad 1 ADDRESS MATCH (ADDRMATCH).

```
*
*AZZERA IL FLAG DI TESTA, IL FLAG ADDRESS, IL CONTATORE
* DELL'INDIRIZZO ALL'INIZIO
*
HFLAG = 0
ADDRMATCH = 0
ADDRCTR = 0
*
*ATTENDE IL CARATTERE DI TESTA, INDIRIZZO DI DESTINAZIONE E
* DI CODA
esegui finchè HFLAG = 0 OPPURE CHAR≠ TRAILER O ADDRCTR≠ 3
*
*ACCETTA CARATTERE SE PRONTO
*
  se RSTB = ATTIVO allora
     RSTB = INATTIVO
     CHAR = INGRESSO
*
*CONTROLLO PER L'INDIRIZZO DEL TERMINALE E CARATTERE DI TESTA
*
  se HFLAG = 1 ED ADDRCTR≠ 3 allora
  se CHAR≠ TERMADDR (ADDRCTR) allora ADDRMATCH = 1
     ADDRCTR = ADDRCTR + 1
  se CHAR = CARATTERE DI TESTA allora HFLAG = 1
  fine
```

Il programma deve ora attendere un carattere di testa, un codice di identificazione di tre digit ed un carattere di coda. Occorre fare attenzione a cosa succede durante l'iterazione quando il programma trova il carttere di testa e cosa succede se un carattere del codice di identificazione errato risulta uguale al carattere di coda.

Un'ulteriore aggiunta può immagazzinare il messaggio in MESSG. NMESS è il numero di caratteri contenuti nel messaggio; se questo alla fine è zero il programma sa che il terminale ha ricevuto un messaggio valido. Non si è cercato di minimizzare le espressioni logiche di questo programma.

```
*
*AZZERA I FLAG, CONTATORI ALL'INIZIO
*
HFLAG = 0
ADDRMATCH = 0
ADDRCTR = 0
NMESS = 0
*
*ATTENDI IL CARATTERE DI TESTA, INDIRIZZO DI DESTINAZIONE
 esegui finchè HFLAG = 0 oppure CAR≠ CODA oppure ADDRCTR≠ 3
*ACCETTA CARATTERE SE PRONTO
*
  se RSTB = ATTIVO allora
     RSTB = INATTIVO
     CHAR = INGRESSO
*
*LEGGI MESSAGGIO SE INDIRIZZO DESTINAZIONE = INDIRIZZO TERMINALE
*
  se HFLAG = 1 ED ADDRCTR = 3 allora
     se ADDRMATCH = 0 E CHAR = CARATTERE DI CODA allora
       MESSG (NMESS) = CHAR
       NMESS = NMESS + 1
*
*CONTROLLO PER L'INDIRIZZO DEL TERMINALE
*
  se HFLAG = 1 ED ADDRCTR = 3 allora
     se CHAR≠ TERMADDR (ADDRCTR) allora ADDRMATCH = 1
     ADDRCTR = ADDRCTR + 1
*
*OSSERVA IL CARATTERE DI TESTA
*
  se CHAR = CARATTERE DI TESTA allora HFLAG = 1
  fine
```

Il programma controlla il codice di identificazione se è stato trovato un carattere di testa durante un'iterazione precedente. Esso accetta il messaggio se ha trovato precedentemente un carattere di testa ed un corrispondente indirizzo di destinazione completo. Il programma deve lavorare correttamente durante l'iterazione quando trova il carattere di testa, quello di coda e l'ultimo digit dell'indirizzo di destinazione. Esso non deve confrontare il carattere di testa con l'indirizzo del terminale oppure posizionare il carattere di coda oppure il digit finale all'indirizzo di destinazione del messaggio. Si può provare ad aggiungere il resto della logica dal diagramma di flusso (Figura 13—13) al programma strutturato. Si noti che l'ordine delle operazioni è spesso critico. Occorre essere sicuri che il programma non completa una fase ed inizia la successiva durante la stessa iterazione.

ANALISI DELLA PROGRAMMAZIONE STRUTTURATA

La programmazione strutturata conduce alla disciplina del progetto del programma. Essa forza a limitare i tipi di strutture che si impiegano e la sequenza delle operazioni. Essa fornisce strutture ad ingresso ed uscita singoli che si possono controllare con precisione logica. Spesso la

programmazione strutturata rende consapevole il progettista delle inconsistenze o possibili combinazioni di ingressi. La programmazione strutturata non è una panacea ma porta qualche ordine in un processo che può essere caotico. La programmazione strutturata potrebbe anche aiutare nella verifica, convalida e documentazione.

La programmazione strutturata non è semplice. Il programmatore deve non solo definire adeguatamente il problema ma deve anche esaminare attentamente la logica. Questo è tedioso e difficile ma può risolversi in una chiara scrittura dei programmi di lavoro.

INDICATORI DI FINE DELLE STRUTTURE

Le strutture particolari non sono ideali e sono spesso inopportune. Inoltre può essere difficile distinguere dove una struttura termina ed un'altra comincia in modo particolare se sono nidificate. In futuro i teorici possono fornire le migliori strutture oppure i progettisti desiderano aggiungerne altre. Sembrano necessari alcuni tipi di indicatori di fine di ogni struttura poichè l'identificazione non sempre chiarisce la situazione. «Fine» è un indicatore di conclusione logica del ciclo «esegui-finchè». Comunque non esistono indicatori di fine per lo statement «se-allora-oppure»; alcuni teorici hanno suggerito «fine dell'if» oppure «fi» («if» alla rovescia) ma entrambi sono inopportuni e non danno contributo alla leggibilità del programma.

REGOLE DELLA PROGRAMMAZIONE STRUTTURATA

Si suggeriscono le seguenti regole per l'applicazione della programmazione strutturata:

1) Si cominci con la scrittura di un diagramma di flusso di base per aiutare a definire la logica del programma.
2) Si inizi a costruire «se-allora-altrimenti» ed «esegui-finchè». È ben noto che esse costituiscono un set completo cioè qualsiasi programma può essere costruito in termini di queste strutture.
3) Si rediga ogni livello in poco spazio così da conoscere a quale statement appartiene.
4) Si impieghino indicatori di fine per ogni struttura, cioè «fine» per l'«esegui-finché» e «fine dell'if» ovvero «fi» per il «se-allora-altrimenti». Gli indicatori di fine assieme alle appartenenze dovrebbero rendere il programma ragionevolmente chiaro.
5) Si accentui la semplicità e la leggibilità. Si usino spazi in abbondanza, si impieghino nomi significativi e si rendano le espressioni più chiare possibile. Non si tenti di minimizzare la logica al costo della chiarezza.
6) Si commenti il programma in maniera organizzata.
7) Si controlli la logica. Si ricerchino tutti i casi estremi e condizioni speciali ad alcuni casi semplici. Qualunque errore trovato a questo livello non affliggerà in seguito.

PROGETTO TOP-DOWN

PROGETTO BOTTOM-UP

Rimane il problema di mostrare il controllo di moduli integrati o strutture. Certamente si dividerà un grosso compito in compiti più piccoli. Ma come controllare isolatamente questi ultimi e poi metterli assieme? La procedura standard chiamata «progetto bottom-up» richiede un lavoro ulteriore nella convalida e verifica e lascia alla fine l'intero compito di integrazione. Si richiede un metodo che consentirà di eseguire la convalida e la verifica ricorrendo al programma effettivo e suddividendo il sistema di integrazione in una serie di compiti modulari.

METODI DI PROGETTO TOP-DOWN

STUB DI PROGRAMMA

Questo metodo è il «progetto top-down». Qui si inizia scrivendo il programma globale di supervisione. Vengono sostituiti i sotto-programmi indefiniti con «stub di programma» (spezzoni di programma) ovvero programmi temporanei che possono registrare l'ingresso, fornire la risposta ad un problema di prova selezionato oppure non eseguire nulla. Poi si prova il programma superiore per vedere se la sua logica è corretta.

Si proseguirà espandendo gli stub. Ogni stub conterrà spesso compiti inferiori che saranno rappresentati temporaneamente come stub. Questo processo di espansione, verifica e convalida continua finchè tutti gli stub non sono stati sostituiti da programmi di lavoro. Si noti che la convalida e l'integrazione ricorrono ad ogni livello piuttosto che quelli alla fine. Non sono necessari driver speciali o programmi di generazione dati. Si ha una idea chiara della loro posizione esatta nel progetto. Il progetto top-down presuppone la programmazione modulare ed è ben compatibile con la programmazione strutturata.

ESPANSIONE DEGLI STUB

VANTAGGI DEL PROGETTO TOP-DOWN

Gli svantaggi del progetto top-down sono:

SVANTAGGI DEL PROGETTO TOP-DOWN

1) Il progetto globale può non essere compatibile col sistema hardware.
2) Esso può non avvantaggiarsi del software esistente
3) Può essere difficile scrivere uno stub appropriato, in modo particolare se lo stub deve lavorare correttamente in posizioni diverse.
4) Il progetto top-down può risolversi in moduli generalmente non comodi.
5) Errori al livello più alto possono avere effetti catastrofici mentre errori nel progetto button-up sono normalmente limitati ad un modulo particolare.

Nei progetti di larga programmazione è stato dimostrato che il progetto top-down migliora enormemente la produttività del programmatore. Comunque quasi tutti questi progetti sono stati impiegati per il progetto button-up in casi dove il metodo top-down si è risolto in una grande quantità di ulteriore lavoro.

Il progetto top-down è uno strumento pratico che non richiede di essere eseguito agli estremi. Esso fornisce la stessa disciplina per i sistemi di convalida ed integrazione che la programmazione strutturata fornisce per il progetto modulare. Il metodo, comunque, è di applicabilità molto più generale poichè esso in realtà non impiega la logica programmata. Comunque il progetto top-down può risolversi in una realizzazione che non è la più efficiente.

ESEMPI

Risposta ad un Interruttore

Il primo esempio di programmazione strutturata si presta bene al progetto top-down. Il programma era:

PROGETTO TOP-DOWN DEL SISTEMA INTERRUTTORE DISPLAY

```
INTERRUTTORE = APERTO
esegue finchè INTERRUTTORE = APERTO
  LETTURA INTERRUTTORE
  fine
DISPLAY = ACCESO
RITARDO 1
DISPLAY = SPENTO
```

Quasi tutti questi statement sono stub poichè nessuno di questi è pienamente definito. Per esempio cosa significa LETTURA INTERRUTTORE? Se l'interruttore era uguale ad un bit della porta d'ingresso SPORT allora in realtà significa:

```
INTERRUTTORE = SPORT E SMASK
```

dove SMASK ha un bit '1' in una posizione appropriata.

Analogamente, RITARDO 1 in realtà significa (se lo stesso processore fornisce il ritardo):

```
REG = CONTEGGIO
esegue finchè REG ≠ 0
  REG = REG - 1
  fine
```

CONTEGGIO è il numero appropriato per fornire il ritardo di un secondo. La versione dettagliata del programma è:

```
INTERRUTTORE = 0
esegue finchè INTERRUTTORE = 0
  INTERRUTTORE = SPORT E MASCHERA
  fine
DISPLAY = ACCESO
REG = CONTEGGIO
esegue finchè REG ≠ 0
  REG = REG - 1
  fine
DISPLAY = NOT (SPENTO)
```

Sicuramente questo programma è più esplicito e potrebbe essere più facilmente trasformato in istruzioni effettive oppure statement.

Il Caricatore di Memoria Basato sugli Interruttori

PROGETTO TOP-DOWN DEL CARICATORE DI MEMORIA BASATO SUGLI INTERRUTTORI

Questo esempio è più complesso del primo esempio cosicchè occorre procedere sistematicamente. Anche qui il programma strutturato contiene stub.

Per esempio se il pulsante ALTO INDIRIZZO è un bit della porta d'ingresso CPORT «se HIADDRBUTTON = 1» in realtà significa:

1) Ingresso da CPORT.
2) Complemento.
3) AND logico con HAMASK;

dove HAMASK ha un '1' in una posizione di bit appropriata e zeri altrove. Analogamente la condizione «se DATABUTTON = 1» in realtà significa:

1) Ingresso da CPORT.
2) Complemento.
3) AND logico con DAMASK.

Così con gli stub iniziali si potrebbe assegnare il valore al pulsante, per esempio:

```
HIADDRBUTTON = 0
LOADDRBUTTON = 0
DATABUTTON = 0
```

Una serie di programmi di supervisione mostrerebbe che esso assume l'«altrimenti» implicato attraverso le strutture «se-allora-altrimenti» e non legge mai gli interruttori. Analogamente se lo stub fosse:

```
HIADDRBUTTON = 1
```

il programma supervisore permarrebbe nel ciclo «esegue-finchè HIADDRBUTTON = 1" in attesa che il pulsante sia rilasciato. Queste semplici serie controllano tutta la logica.

Ora si può iniziare l'espansione di ogni stub ed osservate se essa produce un ragionevole risultato globale. Si noti come la verifica e convalida procedono in un modo modulare e diretto. Si espande lo stub HIADDRBUTTON = 1 a

```
LEGGE CPORT
HIADDRBUTTON = NOT (CPORT) ED HAMASK
```

Il programma dovrebbe attendere la chiusura del pulsante ALTO INDIRIZZO. Il programma dovrebbe poi posizionare i contenuti degli interruttori sul display. Questa serie controlla la risposta corretta al pulsante ALTO INDIRIZZO.

Si espande poi il modulo pulsante BASSO INDIRIZZO a

```
LEGGE CPORT
HIADDRBUTTON = NOT (CPORT) E LAMASK
```

Con il pulsante BASSO INDIRIZZO nella posizione di chiusura, il programma dovrebbe posizionare i contenuti degli interruttori sul display. Questa sequenza controlla la risposta corretta al pulsante BASSO INDIRIZZO.

Analogamente si può espandere il modulo pulsante DATI e controllare la risposta corretta a quel pulsante. L'intero programma sarà poi verificato in una maniera sistematica.

Quando tutti gli stati sono espansi, gli stadi di codifica, verifica e convalida saranno completi. Naturalmente occorre conoscere esattamente quali risultati ogni stub produrrebbe. Comunque molti errori logici diverranno ovvi ad ogni livello senza ulteriore espansione.

Il Terminale di Transazione

PROGETTO TOP-DOWN DEL TERMINALE DI VERIFICA

Questo esempio, naturalmente, avrà più livelli di dettaglio. Si potrebbe iniziare col seguente programma (Vedere Figura 13-17 per un diagramma di flusso).

```
TASTIERA
ACK = 0
esegui finchè ACK = 0
  TRASMETTE
  RICEVE
  fine
DISPLAY
```

Qui TASTIERA, TRASMETTE, RICEVE e DISPLAY sono gli stub di programma che verranno espansi successivamente. La TASTIERA, per esempio, potrebbe semplicemente posizionare un numero composto di dieci digit in un buffer opportuno.

ESPANSIONE DELLA ROUTINE TASTIERA

Lo stadio successivo all'espansione potrebbe produrre il seguente programma per TASTIERA (Vedere la Figura 13-18 per un diagramma di flusso):

```
VER = 0
esegui finchè VER = 0
  COMPLETO = 0
esegui finchè COMPLETO = 0
  KEYIN
  KEYDS
  fine
VERIFICA
fine
```

Dove VERO = 0 significa che un ingresso non è stato verificato; COMPLETO = 0 significa che l'ingresso è un completo, KEYIN e KEYDS sono i tasti d'ingresso e le routine del display rispettivamente VERIFICA controlla l'ingresso. Uno stub possibile per KEYIN potrebbe semplicemente posizionare un ingresso casuale (proveniente da una tabella o generatore di numeri casuali) nel buffer e porre COMPLETO ad 1.

Si potrebbe continuare analogamente espandendo, verificando e convalidando TRASMETTE, RICEVE e DISPLAY. Si noti che si potrebbe espandere ogni programma mediante un livello cosicchè non si deve eseguire l'integrazione di un intero programma in una sola volta. Si potrebbe procedere alla definizione dei livelli con un certo criterio. Una fase troppo piccola spreca tempo mentre una fase troppo grande rimanda ai problemi dell'integrazione del sistema, cosa che si vuole evitare.

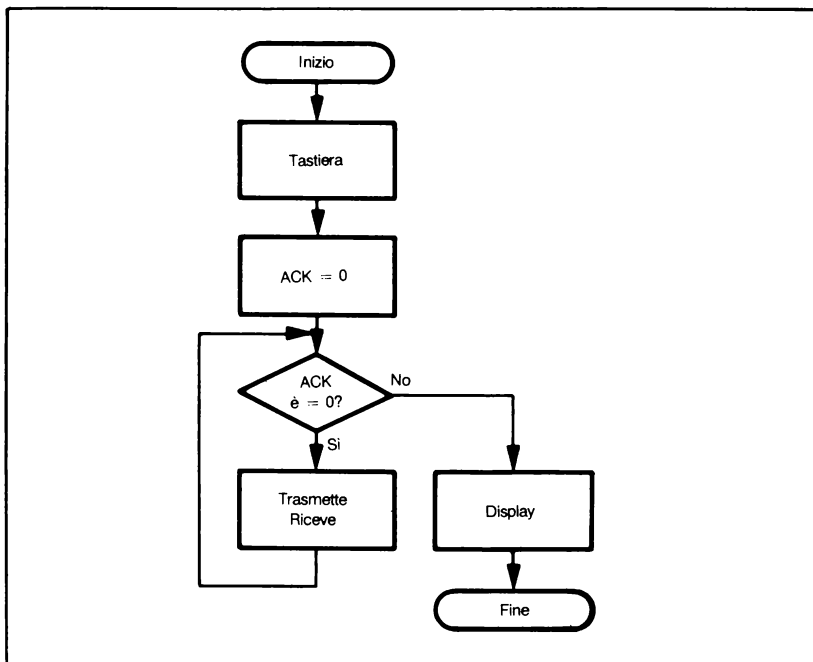



Figura 13-17
Diagramma Di Flusso Iniziale Per Il Terminale Di Operazione

ANALISI DEL PROGETTO TOP-DOWN

Il progetto top - down conduce ad una discpilina delle fasi di verifica ed integrazione del progetto del programma.
Esso fornisce un metodo sistematico per l'espansione di un diagramma di flusso o definizione del problema al livello richiesto per scrivere effettivamente un programma. Assieme alla programmazione strutturata forma un set completo di tecniche di progetto.

Come la programmazione strutturata il progetto top-down non è immediato. Il progettista deve avere accuratamente definito il problema e deve lavorare sistematicamente attraverso ogni li-

vello. Anche qui la metodologia può sembrare tediosa ma i vantaggi possono essere sostanziali se il progettista segue realmente le regole.

FORMATO PER IL PROGETTO TOP-DOWN

Si raccomanda il seguente approccio al progetto top-down:

1) Si inizi con un diagramma di flusso di base.
2) Si eseguano gli stub più completi e separati possibile
3) Si definiscano con precisione tutte le possibili uscite da ogni stub e si selezioni un set di test.
4) Si controlli accuratamente e sistematicamente ogni livello.
5) Si impieghino le strutture della programmazione strutturata.
6) Si espanda ogni stub da un livello. Non si cerchi di fare troppo in una fase.
7) Si valutino attentamente i compiti comuni e le strutture dati.
8) Si esegua il testing e debugging dopo ogni espansione di stub. Non si provi a fare un intero livello alla volta.
9) Ci si renda consapevoli di quale hardware si può fare. Non si esiti ad arrestarsi e fare un piccolo progetto button-up dove necessario.

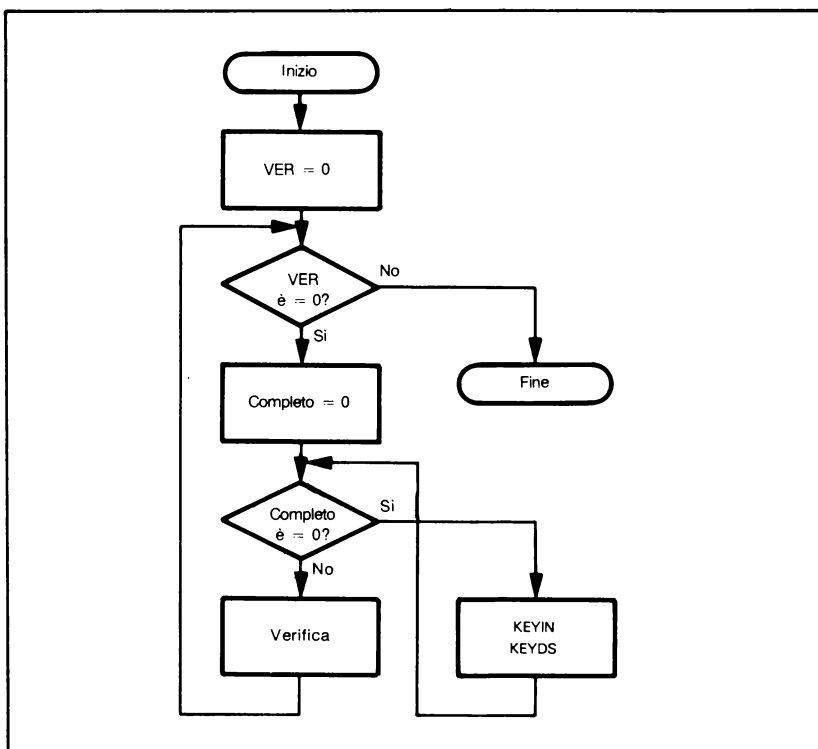



Figura 13-18
Diagramma Di Flusso Per La Routine Dettagliata Della Tastiera

ANALISI DELLA DEFINIZIONE DEL PROBLEMA E PROGETTO DEL PROGRAMMA

Si potrebbe notare che si è speso un intero capitolo senza ricordare nessun microprocessore specifico o linguaggio assembly e senza scrivere una sola riga di codice effettivo. Ora è noto molto di più sugli esempi di cui è stata richiesta inizialmente la scrittura dei programmi. Benchè si pensi spesso alla scrittura delle istruzioni di un calcolatore come una parte chiave dello sviluppo del software, questo è attualmente uno degli stadi più semplici.

Una volta che sono stati scritti diversi programmi la maggior parte della metodologia diviene semplice. Appena appreso il set d'istruzioni, si riconosce che le istruzioni sono veramente semplici e si ricorderanno le sequenze comuni che formano la maggior parte dei programmi. Si troverà poi che molti degli altri stadi dello sviluppo rimangono difficili ed hanno poche regole chiave.

Qui sono stati suggeriti alcuni modi per sintetizzare gli stadi di primaria importanza. Nello stadio di definizione del problema si devono definire le caratteristiche del sistema — i suoi ingressi, uscite, elaborazione, limiti di tempo e memoria e manipolazione degli errori. Occorre considerare in modo particolare come il sistema interagirà con il sistema più grande che comprende un'attrezzatura elettrica, meccanica oppure un operatore umano. A questo stadio occorre iniziare a considerare come rendere il sistema più facile da impiegare e mantenere.

Nello stadio di progetto del programma diverse tecniche possono essere di aiuto a specificare e documentare specificamente la logica del programma. La programmazione modulare forza alla divisione del programma globale in moduli di dimensioni ragionevoli. La programmazione strutturata fornisce un metodo sistematico per la definizione della logica di quei moduli mentre il progetto top-down è un metodo sistematico per integrarli e verificarli. Naturalmente non si é obbligati a seguire tutte queste tecniche: esse sono infatti nient'altro che linee guida. Però esse forniscono un approccio unificato agli stadi definizione e progetto e potrebbero essere considerate come una base di sviluppo per altri approcci.

BIBLIOGRAFIA

1) Hughes, J.L. e J.I. Michtom, «A Structured Approach To Programming», Prentice - Hall, Henglewood Cliffs, N.J. 1977.
2) Leventhal, L.A. «Can Structured Programming Help the Bench Programmer?» IEEE Workshop on Bench Programming of Microprocessors, Philadelphia, P.A. Giugno 1977.
3) Ulrickson, R.W. «Software Modules are the Building Blocks» Electronic Design, 1 Febbraio 1977, pp 62-66.
4) Ulrickson, R.W. «Solve Software Problems Step-by-Step», Electronic Design, 18 Gennaio 1977, pp 54-58.
5) Yourdon, E.U. Tecniques of Program Structure and Design, Prentice - Hall, Englewood Cliffs, N.J. 1975.

Capitolo 14
DEBUGGING E TESTING

Come si è notato all'inizio del precedente capitolo la verifica e la convalida sono tra gli stadi che impiegano più tempo nello sviluppo del software. Anche pensando a metodi come la programmazione modulare, la programmazione strutturata ed il progetto top-down che possono ridurre la frequenza degli errori e semplificare il testing, il debugging ed il testing sono ancora difficoltosi poichè essi sono compiti scarsamente definiti. Molti libri di testo sono stati scritti sulla progettazione e codifica di programmi. Invece pochissimi sono stati scritti su come operare il debugging ed il testing dei programmi. La selezione di un adeguato set di dati di test è raramente un processo chiaro o scientifico. La ricerca degli errori talvolta sembra come un gioco di abilità. Sicuramente pochi compiti sono snervanti come il debugging dei programmi.

Questo capitolo descriverà innanzi tutto gli strumenti disponibili per il debugging. Si discuterà poi la procedura di debugging di base, si descriveranno i tipi più comuni di errore e si mostreranno alcuni esempi di debugging di programmi. Alla fine si descriverà come selezionare i dati ed i programmi di test.

Non si farà molto di più che descrivere lo scopo dei maggiori strumenti di debugging. C'è una ben nota standardizzazione in questo settore e non abbastanza spazio per discutere tùtti i dispositivi ed i programmi disponibili sul mercato. Gli esempi forniranno qualche idea su quando ed in quale particolare hardware e software sarà conveniente.

STRUMENTI SEMPLICI DI DEBUGGING

I più semplici strumenti di debugging disponibili sono:

1) Possibilità di fase singola (single-step)
2) Possibilità di un punto di arresto (breakpoint)
3) Un programma di scarico di registro (od utilità)
4) Un programma di scarico di memoria

SINGOLA FASE

La possibilità di singola-fase permette semplicemente di eseguire il programma ad una fase alla volta. La maggior parte dei microcomputer basati sull'Intel 8080 hanno questa possibilità poichè la circuiteria é abbastanza semplice. Naturalmente quando si esegue una singola fase si può osservare solo gli stati delle linee di uscita che si stanno osservando. Le linee più importanti sono

1) Bus Dati
2) Bus Indirizzi
3) Uscite latch di stato
4) Linee di controllo

Se si osservano queste linee (in hardware oppure in software) si vedranno gli indirizzi, le istruzioni e i dati appariranno non appena il programma li esegue. Si potrà vedere quale tipo di operazioni sta eseguendo la CPU mediante l'esame di queste linee in istanti opportuni. Questa informazione sarà sufficiente ad informare di errori come istruzioni di salto non corrette, indirizzi omessi o non corretti, codici di operazione errati o non corretti, valori dei dati. Comunque non

si possono vedere i contenuti dei registri e dei flag senza alcune possibilità addizionali di debugging oppure una sequenza speciale di istruzioni. Perciò molta logica del programma può rimanere invisibile.

LIMITAZIONI DEL MODO A FASE SINGOLA

Ci sono molti errori che un modo a fase singola non può trovare. Questi comprendono errori di timing od errori dei sistemi di interrupt oppure DMA. Inoltre il modo a fase singola è molto lento, tipicamente opera ad un milionesimo della velocità del processore stesso. Perciò, per un secondo del tempo reale del processore, con il modo a fase singola si potrebbe impiegare più di dieci giorni. Chiaramente il metodo a fase singola è conveniente solo per controllare la logica di sequenze di istruzioni corte.

PUNTO DI ARRESTO

Un punto di arresto (breakpoint) è un indirizzo al quale automaticamente si arresterà la sua normale routine così da esaminare lo stato attuale del sistema. Normalmente il programma non ripartirà ancora finché si azzera il punto di arresto. Il punto di arresto permette di controllare o di passare attraverso un'intera sezione di un programma. Così per vedere se una intera routine di inizializzazione è corretta si può posizionare un punto di arresto alla fine dell'inizializzazione e far partire il programma. Si può poi controllare lo stato delle locazioni di memoria ed i registri per vedere se la intera sezione è corretta.

I punti di arresto sono il complemento del modo a fase singola. Si può impiegare il punto di arresto sia per localizzare errori o per passare attraverso sezioni che si conosce essere corrette. Si può poi eseguire una verifica dettagliata nel modo a fase singola. I punti di arresto normalmente non influenzano il timing del programma cosicché possono essere impiegati per controllare l'ingresso/uscita e gli interrupt.

RST COME UN PUNTO DI ARRESTO

I punti di arresto spesso si avvantaggiano di parte o tutto il sistema di interrupt del microprocessore. Alcuni hanno la possibilità di INTERRUPT SOFTWARE oppure TRAPPOLA che può agire come punto di arresto. Sull'Intel 8080 se non si stanno impiegando tutti i vettori di interrupt nel programma si può impiegare l'istruzione RST (restart: riparti) come punto di arresto. La Tabella 14-1 dà l'indirizzo di destinazione per le varie istruzioni RST. Il capitolo 12 descrive, con maggior dettaglio, l'istruzione RST. La routine del punto di arresto può stampare i contenuti dei registri e della memoria oppure attendere in quella posizione (HLT oppure un salto condizionato o incondizionato all'istruzione stessa) finchè l'utente non permette alla macchina di procedere. Si ricordi però che l'istruzione RST impiega lo Stack ed il Puntatore dello Stack per immagazzinare l'indirizzo di ritorno. La Figura 14-1 mostra una routine dove RST3 si risolve in un loop senza fine. Il programmatore dovrebbe azzerare questo punto di arresto con un RESET oppure un segnale di interrupt.

```
        ORG     18H
RST3    EQU     18H
        IMP     RST3     ;ATTENDE IN POSIZIONE
```

Figura 14-1
Una Semplice Routine Con Punto Di Arresto

Una possibilità più potente potrebbe permettere di entrare all'indirizzo a cui il processore dovrebbe trasferire il controllo. Un'altra possibilità potrebbe essere un ritorno dipendente su un interruttore:

```
        ORG     18H
RST3    EQU     18H
        PUSH    PSW     ;SALVA A
WAITS:  IN      SPORT   ;ATTENDE CHE INTERRUTTORE = 1
        ANI     MASK
        JZ      WAITS
        POP     PSW     ;RI-IMMAGAZZINA A
        RET
```

I programmi monitor e dei sistemi di sviluppo generalmente forniranno diversi tipi di possibilità pratiche di punti di arresto.

Istruzione (Mnemonico)	(Esadec.)	Indirizzo di Dest. (Esadec.)
RST 0	C7	0000
RST 1	CF	0008
RST 2	D7	0010
RST 3	DF	0018
RST 4	E7	0020
RST 5	EF	0028
RST 6	F7	0030
RST 7	FF	0038

Tabella 14-1
Indirizzi Delle Istruzioni Di Risposta Per L'Intel 8080

L'utilità dello scarico di un registro su un microcomputer è un programma che mostrerà i contenuti di tutti i registri della CPU. La seguente routine mostrerà i contenuti dei registri se si assume che STAMPA mostri i contenuti dell'Accumulatore come due digit esadecimali.

La Figura 14-2 è un diagramma di flusso del programma e la Figura 14-3 mostra un risultato tipico.

```
;
;POSIZIONA TUTTI I CONTENUTI NELLO STACK
;
        PUSH    PSW     ;PC GIÀ NELLO STACK
        PUSH    B
        PUSH    D       ;CONSERVA I REGISTRI NELLO STACK
        PUSH    H
;
;IMPIEGA IL PUNTATORE DELLO STACK COME INDIRIZZO DI PARTENZA
;
        LXI     H,0
        DAD     SP      ;ACCETTA IL PUNTATORE DELLO STACK
        LXI     D,10    ;COMPLETA IL VALORE DELL'SP ORIGINARIO
        DAD     D
        PUSH    H       ;CONSERVA IL PUNTATORE DELLO STACK ORIGINARIO

;STAMPA I CONTENUTI DEI REGISTRI
;L'ORDINE È PC (ALTO), PC (BASSO), A, FLAG, B, C, D, E, H, L, SP
;(ALTO), SP (BASSO)

        MVI     C,12    ;NUMERO DI BYTE = 12
PRN1:   DCX     H
        MOV     A,M     ;ACCETTA UN BYTE DALLO STACK
        CALL    PRNT    ;E LO STAMPA
        DCX     H
        DCR     C
        JNZ     PRN1
;
;RI-IMMAGAZZINA I REGISTRI DALLO STACK
;
        POP     H       ;RI-IMMAGAZZINA I REGISTRI DALLO STACK
        POP     H
        POP     D
        POP     B
        POP     PSW
        RET
```

Si noti che la chiamata della routine di scarico di registro posiziona il vecchio valore del Contatore di Programma nello Stack.

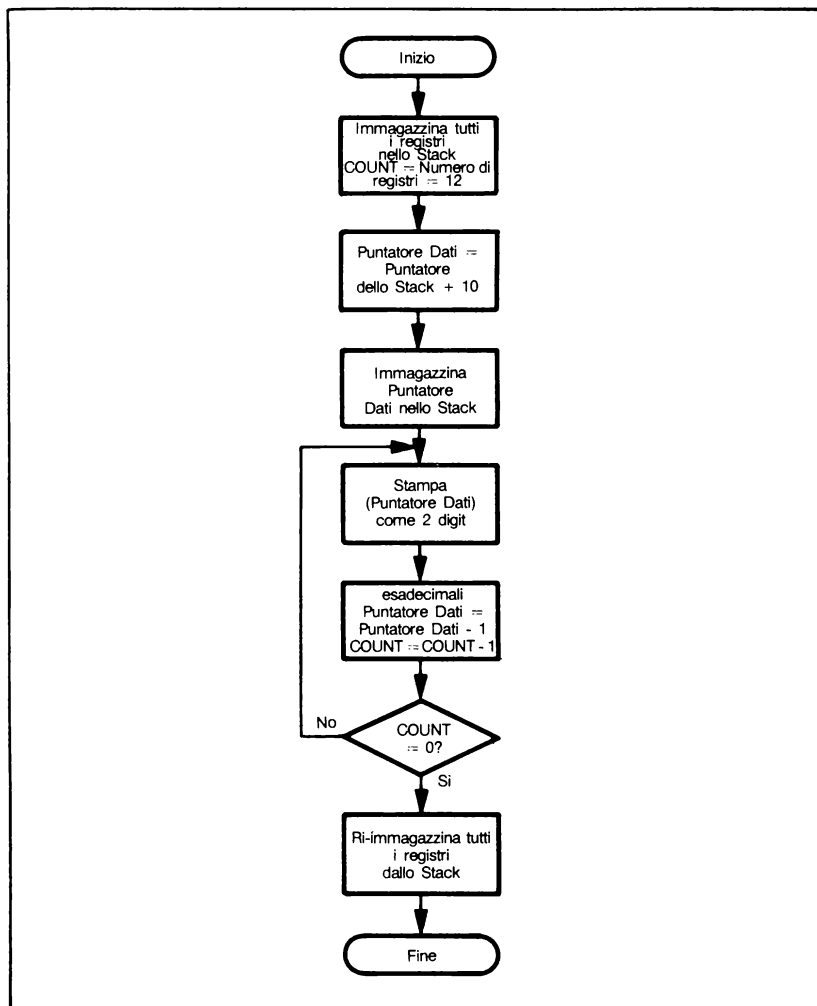



Figura 14-2
Diagramma Di Flusso Del Programma Di Scarico Di Registro

3E	(CONTATORE DI PROGRAMMA)
FC	
86	(PSW)
1D	(A)
07	(B)
3E	(C)
23	(D)
01	(E)
17	(H)
F0	(L)
FC	(PUNTATORE DELLO STACK)
3F	

Figura 14-3
Risultati Di Un Tipico Scarico Di Registro

SCARICO DI MEMORIA

Uno scarico di memoria è un programma che può mostrare i contenuti di un intera sezione di memoria. Questo è molto più efficiente per l'esame di un array o blocchi piuttosto che singole locazioni. Comunque scarichi di memoria molto grandi non sono generalmente comodi a causa della grande massa di informazione che producono. Esse impiegano anche molto tempo per essere generate su una stampante lenta. Comunque piccoli scarichi possono fornire al programmatore una ragionevole quantità di informazioni che possono essere esaminate a parte. Relazioni come ripetizioni regolari di strutture dati oppure gli offset di interi array possono diventare ovvi. Uno scarico generale può risultare piuttosto difficile da scrivere. Il programmatore dovrebbe prestare attenzione alle seguenti situazioni:

1) La dimensione dell'area di memoria eccede 256 byte cosicchè un singolo contatore non è sufficiente.
2) La locazione terminale ha un indirizzo più piccolo della locazione iniziale. Questo potrebbe essere trattato come un errore di memoria in un ordine particolare.

1000	23	1F	60	54	37	38	3E	00
1008	6E	43	38	17	59	44	98	37
1010	47	36	23	81	E1	FF	FF	5A
1018	34	ED	BC	AF	FE	FF	27	02

Figura 14-4
Risultati Di Un Tipico Scarico Di Memoria

Poichè la velocità dello scarico di memoria è normalmente dipendente dalla velocità della stampante, che è sempre più lenta della velocità del processore, raramente interessa l'efficienza della routine. Il seguente programma rigetterà i casi dove l'indirizzo iniziale è dopo l'indirizzo finale e manipolerà blocchi di qualsiasi lunghezza. Si assume che l'indirizzo iniziale è nei Registri H ed L e quello finale nei Registri D ed E:

```
;
;OSSERVA SE LA FINE È PRIMA DELL'INIZIO
;
        MOV     A,D     ;ESAMINA GLI MSB DELL'INDIRIZZO
        CMP     H
        JC      DONE    ;ATTRAVERSA SE L'INIZIO È MAGGIORE
        JNZ     DUMP    ;VA BENE SE L'INIZIO È MINORE
        MOV     A,E     ;SE GLI MSB SONO UGUALI, OSSERVA GLI LSB
        CMP     L
        JC      DONE    ;ATTRAVERSA SE L'INIZIO È MAGGIORE
;STAMPA LA LOCAZIONE 1
;
DUMP:   MOV     A,M     ;ACCETTA LA LOCAZIONE 1
        CALL    PRNT    ;E STAMPA LA
        MOV     A,D     ;ESAMINA GLI MSB DELL'INDIRIZZO
        CMP     H
        JNZ     CONT    ;CONTINUA SE NON UGUALE AL LIMITE
        MOV     A,E     ;ESAMINA GLI LSB DELL'INDIRIZZO
        CMP     L
        JZ      DONE    ;FATTO SE AL LIMITE
CONT:   INX     H
        JMP     DUMP
DONE:   JMP     DONE
```

La Figura 14-4 mostra l'uscita da uno scarico delle locazioni di memoria da 1000 a 101F. Questa routine manipola correttamente il caso in cui le locazioni d'inizio e di fine sono la stessa. L'utente dovrà accuratamente esaminare i risultati se lo scarico dell'area coinvolge lo Stack poichè la subroutine di scarico impiega essa stessa lo stack. PRNT può anche cambiare parte della memoria.

STRUMENTI DI DEBUGGING PIÙ AVANZATI

Gli strumenti di verifica più avanzata più largamente impiegati sono:

1) Programmi simulatori per controllo software.
2) Analizzatori logici per controllare i segnali ed il timing.

Esistono molte variazioni ad entrambi questi strumenti e si discuteranno le caratteristiche standard.

SIMULATORE

Il simulatore è l'equivalente computerizzato del computer carta-e-matita. Consiste di un programma del computer che fa partire il ciclo di funzionamento di un altro programma del computer mantenendo la traccia dei contenuti di tutti i registri, flag e locazioni di memoria. Naturalmente questo potrebbe essere fatto normalmente, ma richiederebbe uno sforzo ed un'attenzione inaccettabilmente elevati per l'effetto preciso di ogni istruzione. Il programma simulatore, d'altra parte, non si confonde od affatica non dimentica istruzioni od uscite di carta. La maggior parte dei simulatori sono grossi programmi FORTRAN. Essi possono essere acquistati oppure si può richiedere il servizio time-sharing. Il simulatore 8080 è disponibile in diverse versioni da sorgenti diverse.

Le caratteristiche tipiche del simulatore sono:

1) La possibilità di punti di arresto. I punti di arresto generalmente possono essere posti per ricorrervi dopo un particolare numero di cicli quando si fa riferimento ad una locazione di memoria od una di un set di locazioni di memoria, quando i contenuti di una locazione od una di un set di locazioni è alterata o su altre condizioni specifiche del particolare simulatore.
2) Possibilità di scarico di registro e memoria che può mostrare i valori di locazioni di memoria, registri e porte I/O.
3) La possibilità di traccia (trace facility) che stamperà i contenuti di registri o locazioni di memoria particolari, ogni volta che il programma li cambia o vi si riferisce.
4) La possibilità di carico che permette all'utente di imporre i valori iniziali o di cambiarli durante la simulazione.

Alcuni simulatori possono anche simulare ingresso/uscita, interrupt ed anche accesso diretto in memoria.

Il simulatore ha molti vantaggi:

1) Può fornire una descrizione completa dello stato del computer poichè il programma non è ristretto da limitazioni di più od altri problemi.
2) Può fornire punto di arresto, scarico, traccia ed altre possibilità che non coinvolgeranno la memoria del processore. Queste possibilità perciò non interferiranno con il programma utente.
3) Programmi, punti d'inizio ed altre condizioni sono facili da cambiare.
4) Tutte le possibilità dei grandi computer comprendono periferiche e software, sono disponibili al progettista di microprocessori.

D'altra parte, il simulatore è limitato dal suo software di base e la sua separazione dal computer effettivo. Le maggiori limitazioni sono:

1) Il simulatore normalmente non può essere di aiuto per problemi di timing.
2) Il simulatore non può ricreare appieno il sistema d'ingresso/uscita.
3) Il simulatore è normalmente abbastanza lento. La divagazione di un secondo del tempo del processore può impiegare ore del tempo di calcolo. L'impiego del simulatore può essere abbastanza dispendioso.

Il simulatore rappresenta la parte software del debugging; esso ha i vantaggi e le limitazioni tipiche di un approccio completamente di tipo software. Fondamentalmente il simulatore può fornire penetrazione nella logica del programma ed altri problemi software ma non può aiutare con timing, I/O, ed altri problemi hardware.

ANALIZZATORE LOGICO

L'analizzatore logico o di microprocessore è la soluzione hardware per la verifica. (Per informazioni più complete vedere Runyon, «Focus on Logic and μP Analyzers» Electronic Design, 1 Febbraio 1977, pp 40-50). Fondamentalmente l'analizzatore è una versione digitale parallela dell'oscilloscopio standard. L'informazione è mostrata in binario dall'analizzatore oppure in esadecimale su un display video ed ha una varietà di eventi di trigger, soglie ed ingressi. La maggior parte degli analizzatori è dotata di memoria così da poter mostrare i contenuti passati dei bus.

La procedura standard è di imporre un evento di trigger come all'occorrenza di un particolare indirizzo sul Bus degli Indirizzi od istruzione sul Bus dei Dati in un indirizzo particolare oppure nell'esecuzione di un'istruzione di ingresso od uscita. Si può poi osservare la sequenza di eventi che procede o segue il punto di arresto. Nei problemi più comuni si può trovare che questo modo coinvolge punte di rumore (glitche), sequenze di segnale incorretto, sovrapposizione di forme d'onda ed altri errori di timing o segnale. Naturalmente il simulatore software non può fornire nessun suggerimento sull'esistenza di quegli errori di cui molto di più un analizzatore logico potrebbe effettivamente mettere in guardia il programmatore da errori nella logica del programma.

L'analizzatore logico può anche essere comodo nel controllo del funzionamento corretto del microcomputer (Vedere Farnbach, W.A., «Bring up Your μP» Electronic Design, 19 luglio 1976, pp 80-85).

CARATTERISTICHE IMPORTANTI DEGLI ANALIZZATORI LOGICI

Gli analizzatori logici variano in molti aspetti. Alcuni di questi sono:

1) Numero delle linee d'ingresso. Ne servono almeno 24 per osservare un Bus Dati ad 8 Bit ed un Bus degli Indirizzi a 16 bit. Ancora di più ne servono per i segnali di controllo, clock, ed altri ingressi importanti.
2) Quantità di memoria. Ogni stato che è conservato richiederà diverse parole.
3) Frequenza massima. Essa deve essere diversi MHz per manipolare i processori più veloci.
4) Larghezza minima degli impulsi di segnale (importante per rivelare glitche).
5) Tipo e numero degli eventi di trigger consentiti.
6) Metodi di connessione al microcomputer. Questo può richiedere un'interfaccia piuttosto complessa.
7) Numero di canali display.
8) Display binari, esadecimali o mnemonici.
9) Formati di display.
10) Possibilità di campionamento nel tempo del segnale.
11) Capacità della sonda.
12) Soglie singole o doppie, soglie regolabili.

Tutti questi fattori sono importanti nella valutazione di analizzatori logici e di microprocessori poichè l'intero settore è nuovo e non standardizzato. Una grande varietà di prodotti è già disponibile e questo settore si espanderà in futuro.
Naturalmente gli analizzatori logici sono necessari soltanto per sistemi con timing complesso. Applicazioni semplici, con periferiche a bassa velocità, hanno alcuni problemi hardware che il progettista non può manipolare con un oscilloscopio standard.

DEBUGGING CON LISTE DI CONTROLLO

Naturalmente il progettista non può controllare manualmente un intero programma. Però esistono alcuni punti critici che il progettista può facilmente controllare. Si può impiegare il controllo manuale sistematico per trovare un grande numero di errori senza ricorrere a nessuno strumento di debugging.

COSA INCLUDERE NELLE LISTE DI CONTROLLO

Il problema è dove posizionare lo sforzo. La risposta è nei punti che possono essere manipolati con delle risposte sì-no o con semplici calcoli aritmetici. Non si cerchi di operare sull'aritmetica complessa, seguita da tutti i flag, oppure di provare ogni caso concepibile. Si limiti il controllo manuale ad elementi che possono essere facilmente posizionati. Lasciare gli elementi più complessi di vari strumenti di debugging. Ci si assicuri, comunque, di procedere sistematicamente, si costruisca la lista di controllo (check list) e ci si assicuri che il programma esegue correttamente l'elaborazione di base.

La prima cosa da fare è confrontare il diagramma di flusso o programma strutturato con il codice attivo. Ci si assicuri che ogni elemento che compare in uno appaia anche nell'altro. Una semplice lista di controllo eseguirà il lavoro. È facile omettere semplicemente un ramo di una sezione di elaborazione.

Successivamente ci si concentri sui cicli di programma. Ci si assicuri che tutti i registri e le locazioni di memoria impiegati all'interno del loop siano state inizializzate prima di essere impiegate. Questa è una sorgente comune di errori; infine è sufficiente una semplice lista di controllo.

Si osservi ora ogni ramo condizionale. Si selezioni un caso campione che produce un ramo ed uno no; si verifichino entrambi. Il ramo è corretto o invertito? Se il ramo coinvolge un controllo se un numero è sopra o sotto una soglia, si provi i casi di uguaglianza. Il ramo segue la via corretta? Ci si assicuri che la scelta è consistente con la definizione del problema.

Si osservi il ciclo come tutt'uno. Si provino manualmente la prima e l'ultima iterazione. Queste spesso sono casi particolarmente preoccupanti. Cosa succede se il numero di iterazione è zero, cioè se non ci sono dati oppure la tabella non ha elementi? Il programma lavora correttamente? I programmi spesso inizieranno eseguendo un'iterazione del ciclo prima di controllare se il compito è completo oppure, ancora peggio, decrementa sotto zero i contatori prima di controllarli. Si controlli ogni cosa sotto l'ultimo statement. Non si assuma (fiduciosamente) che il primo errore è il solo del programma. Il controllo manuale consentirà di ottenere il massimo beneficio dalla verifica di liberarsi molti piccoli errori guadagnando in tempo.

DOMANDE SUL CONTROLLO MANUALE

Un'analisi veloce delle domande del controllo manuale è:

1) Ogni elemento del progetto del programma è contenuto nel programma (e viceversa per scopo di documentazione)?
2) Tutti i registri e locazioni di memoria impiegati all'interno dei cicli sono inizializzati prima di essere impiegati?
3) Tutti i rami condizionali sono corretti?
4) Tutti i cicli iniziano e terminano correttamente?
5) I casi di uguaglianza sono manipolati correttamente?
6) I casi banali sono manipolati correttamente?

OSSERVAZIONE DEGLI ERRORI

Naturalmente, nonostante tutte queste precauzioni (oppure se non si tiene conto di alcune di esse) i programmi quasi mai lavorano la prima volta che vengono eseguiti. Al progettista rimane quindi il problema di dove iniziano ad indagare per l'errore. La lista di controllo manuale fornisce una posizione di partenza se essa non è stata usata precedentemente; alcuni degli errori che possono non essere stati eliminati sono:

ERRORI COMUNI

1) Errori di inizializzazione di variabili come contatori, puntatori, somme, ecc.. Non si assuma che nulla è necessariamente zero quando si inizia.
2) Inversione della logica di un salto condizionato, per esempio impiegando un Salta-Su-Carry quando si richiede Salta-Su-Non-Carry. Si ricordino gli effetti di un confronto e sottrazione:
   (A sono i contenuti dell'Accumulatore, M del registro o locazione di memoria).

$$\begin{aligned} \text{ZERO} &= 1 \text{ se } A = M \\ &= 0 \text{ se } A \neq M \\ \text{CARRY} &= 1 \text{ se } A < M \\ &= 0 \text{ se } A \geq M \end{aligned}$$

In particolare si noti che CARRY = 0 se A = M cioè nel caso di eguaglianza. Così, Salta-Su-Carry significa salta se A < M Salta-Su-Non-Carry significa salta se A $\geq$ M. Se si vuole il caso di eguaglianza dall'altra parte si provi di invertire i ruoli di A ed M oppure di aggiungere 1 ad M. Per esempio se si vuole un salto se A $\geq$ 10 si impieghi:

```
CPI     10
JNC     ADDR
```

Se, d'altra parte, si vuole un salto se A < 10 si impieghi:

```
CPI     11
JNC     ADDR
```

3) Aggiornamento dei contatori e puntatori nella posizione sbagliata od omissione totale. Ci si assicuri che non ci siano traiettorie attraverso un ciclo che saltano o si ripetono un'istruzione di aggiornamento.

4) Errori in casi banali come dati non presunti in un buffer, nessun test da eseguire, nessun ingresso in una transizione. Non si assuma che questi non si verificheranno mai finchè il programma non li ha specificatamente eliminati.
Altri problemi da osservare sono:

5) Inversioni dell'ordine degli operandi. Si ricordi che l'istruzione MOV muove il secondo operando nel primo. Per esempio MOV B, A muove A a B e non viceversa.

6) Variazioni delle condizioni dei flag prima di usarli. Si ricordi che INR e DCR influenzano tutti i flag eccetto il Carry. Si ricordi anche che POP PSW cambia tutti i flag le istruzioni logiche azzerano Carry.

7) Errori nel cambiare i flag di condizione.
I flag Zero o Sign possono non rappresentare lo stato corrente dell'Accumulatore poichè molte istruzioni (per esempio MOV, LDA, MVI) non cambiano i flag.
Si ricordi anche che INX, DCX e CMA influenzano non completamente i flag.

8) Confusione di valori ed indirizzi.
Si ricordi che LXI H, 1000H carica HL con il numero 1000 (esadec.) mentre LHLD 100H carica HL con i <u>contenuti</u> delle locazioni 1000 e 1001. Una distinzione simile si applica ad LDA COUNT ed MVI A,COUNT.

9) Ri-inizializzazione accidentale di un registro o locazione di memoria.
Ci si assicuri che gli indirizzi di salto siano posizionati in modo che non vengano eseguite ulteriori inizializzazioni.

10) Confusione di numeri e caratteri.
Si ricordi che le rappresentazioni ASCII dei digit differiscono dai digit stessi, per esempio, ASCII 7 è 37 esadecimale, 07 è il carattere campanello ASCII.

11) Confusione di binario e decimale.
Si ricordi che la rappresentazione BCD di un numero è diversa dalla sua rappresentazione binaria, per esempio BCD 36 è rappresentato dal numero binario 00110110. Si noti che il decimale equivalente di questo numero binario è 54 e non l'originale BCD 36.

12) Esecuzione di sottrazioni non bene allineate. Ci si assicuri anche per altre operazioni (come divisione) che non commutano.
Si ricordi che SUB M e CMP M producono A-M e non M-A.

13) Mancata conoscenza degli effetti di subroutine e macro. Non si assuma che la chiamata alla subroutine oppure i riferimenti ai macro non varino i flag, registri o locazioni di memoria. Ci si assicuri di quale effetto essi hanno esattamente. Si noti che è molto importante documentare questi effetti.

14) Impiego improprio di istruzioni di spostamento.
Si ricordino gli effetti precisi di RAR, RLC ed RRC. Esse sono tutte istruzioni di scorrimento circolare di 1-bit che influenzano solo il Carry. Si noti che non esiste uno scorrimento aritmetico (con prescrizione del bit segno) e neppure lo spostamento logico a destra (ADD A è uno scorrimento logico a sinistra).

15) Conteggio non corretto della lunghezza di un array.
Si ricordi che esistono cinque, non quattro, locazioni di memoria comprese negli indirizzi da 100 a 104 compresi.

16) Dimenticanza che certi trasferimenti impiegano sempre l'Accumulatore.
Questi sono LDA, STA, IN, OUT, LDAX e STAX. Si impieghi particolare attenzione con le ultime due che usano la coppia specifica di registri come indirizzo ed usano l'Accumulatore come sorgente o destinazione dei dati. Per esempio LDAX B carica l'Accumulatore con i dati provenienti dall'indirizzo contenuto nella coppia di registri B.

17) Confusione di registri e coppie di registri.
Si ricordi che LXI, LHLD, DCX, INX, DAD, SHLD, POP e PUSH influenzano coppie di registri e non singoli registri. D'altra parte MVI, MOV, DCR ed INR influenzano singoli registri.

18) Dimenticanza di inizializzazione dello Stack.
Si ricordi che si deve inizializzare il Puntatore dello Stack prima della chiamata di qualunque subroutine od esecuzione di qualunque operazione sullo stack.

19) Cambiamento di un registro o locazione di memoria prima di impiegarlo.
Si ricordi che LDA, MOV, MVI ecc. cambiano i contenuti della destinazione (ma non la sorgente).

DEBUGGING DI PROGRAMMI GUIDATI DA INTERRUPT

I programmi guidati da interrupt sono particolarmente difficili al debugging poichè gli errori possono ricorrere casualmente. Se, per esempio, il programma abilita gli interrupt anticipando di alcune istruzioni, ricorrerà un errore se è ricevuto un interrupt mentre il programma sta eseguendo quelle istruzioni. Infatti si può normalmente assumere che gli errori ricorrenti casualmente siano originati dal sistema di interrupt. (Vedere Weller, U.J., Assembly Level Programming For Small Computers. Lexington Books, Lexington, Mass. 1975).

Errori tipici dei programmi guidati da interrupt sono:

1) Dimenticanza di riabilitare l'interrupt dopo averne accettato e servito uno.
Il processore disabilita automaticamente il sistema di interrupt su RESET od accettando un interrupt. Ci si assicuri che non siano possibili sequenze errate che abilitino il sistema di interrupt.

2) Impiego dell'Accumulatore prima di averlo conservato; cioè PUSH PSW deve venire prima di qualunque operazione d'ingresso od uscita.

3) Dimenticanza di conservare l'Accumulatore ed i flag.

4) Ri-immagazzinamento dei registri in un ordine errato.
Se l'ordine in cui essi erano conservati era:
```
PUSH  PSW
PUSH  B
PUSH  D
PUSH  H
```
l'ordine di ri-immagazzinamento dovrebbe essere
```
POP  H
POP  D
POP  B
POP  PSW
```

5) Abilitazione di interrupt prima di avere stabilito tutte le condizioni necessarie come priorità, flag, ecc.
Qui può essere di aiuto una lista di controllo.

6) Permanenza in registri dei risultati e loro distruzione nel processo di ri-immagazzinamento.

7) Dimenticanza che RST lascia un indirizzo nello Stack sia che si usi o no.
Si può ri-inizializzare oppure aggiornare il Puntatore dello Stack.

8) Non disabilitazione dell'interrupt durante trasferimenti a parola multipla o sequenze di istruzione.

Fiduciosamente almeno queste liste forniranno qualche idea di dove indagare. Sfortunatamente anche la verifica più sistematica lascerà ancora qualche problema veramente imbarazzante.

ESEMPI DI DEBUGGING

Conversione da Decimale a 7-Segmenti

DEBUGGING DI UN PROGRAMMA DI CONVERSIONE DI CODICE

Il programma converte il numero decimale della locazione di memoria 40 in un codice a 7 segmenti nella locazione 41. Esso azzera il display se la locazione 40 non contiene un numero decimale.

Programma Iniziale: (Dal Diagramma di flusso della Figura 14-5)

```
        LDA     40H     ;ACCETTA DATI
        CPI     9
        JC      DONE    ;SALTA SE DATI >9
        LHLD    SSEG    ;ACCETTA INDIRIZZO DELLA TABELLA A 7-SEGMENTI
        MOV     D,A
        DAD     D       ;TROVA ELEMENTO MEDIANTE INDICIZZAZIONE
        MOV     A,M
DONE:   STA     41H     ;ACCETTA IL CODICE A 7 SEGMENTI
HERE:   JMP     HERE
SSEG:   DB      3FH, 06H, 5BH, 4FH, 69H
        DB      6DH, 70H, 07H, 7DH, 6FH
```

Impiegando la procedura delle liste di controllo si è in grado di trovare i seguenti errori:

1) È stato omesso il blocco che azzera il RISULTATO.
2) La diramazione condizionale era non corretta.

Per esempio se i dati sono 00, CPI 9 pone ad uno il Carry poichè $0 < 9$. Comunque il salto nella condizione opposta, cioè JNC DONE, non produrrà ancora il risultato corretto. Ora il caso di uguaglianza è sbagliato poichè, se il dato era 9, CPI 9 azzera il Carry ed origina un salto.

La versione corretta è

```
        CPI     10
        JNC     DONE    ;SALTA SE DATI >9
```

Secondo Programma:

```
        MVI     B,0     ;AZZERA DISPLAY
        LDA     40H     ;ACCETTA DATI
        CPI     10
        JNC     DONE    ;ATTRAVERSA SE DATI >9
        LHLD    SSEG    ;ACCETTA INDIRIZZO DELLA TABELLA A 7 SEGMENTI
        MOV     D,A
        DAD     D       ;TROVA INGRESSO TABELLA MEDIANTE INDICIZZAZIONE
        MOV     A,M     ;ACCETTA IL CODICE A 7 SEGMENTI
DONE:   STA     41H
HERE:   JMP     HERE
SSEG:   DB      3FH, 06H, 5BH, 4FH, 69H
        DB      6DH, 7DH, 07H, 7DH, 6FH
```

Questa versione è stata successivamente controllata manualmente. Poichè il programma è semplice lo stadio successivo consiste di un attraversamento a fase singola con dati effettivi. I dati selezionati per la prova sono:

0	(il numero più piccolo)
09	(il numero più grande)
10	(un caso estremo)
6B (esadec.)	(casuale)

La prima prova è con zero nella locazione 40 (esadec.). Il programma non sviluppa errori apparenti finchè cerca di eseguire l'istruzione MOV A,M.

I contenuti del Bus degli Indirizzi durante l'accettazione dei dati è 0647, un indirizzo che ancora non esiste nel sistema. Chiaramente c'è qualcosa di errato.

È quindi il caso di eseguire più controlli manuali. Poichè si conosce che JNC opera correttamente l'errore è chiaramente successivo a quella istruzione ma prima di MOV A,M. Un controllo manuale mostra:

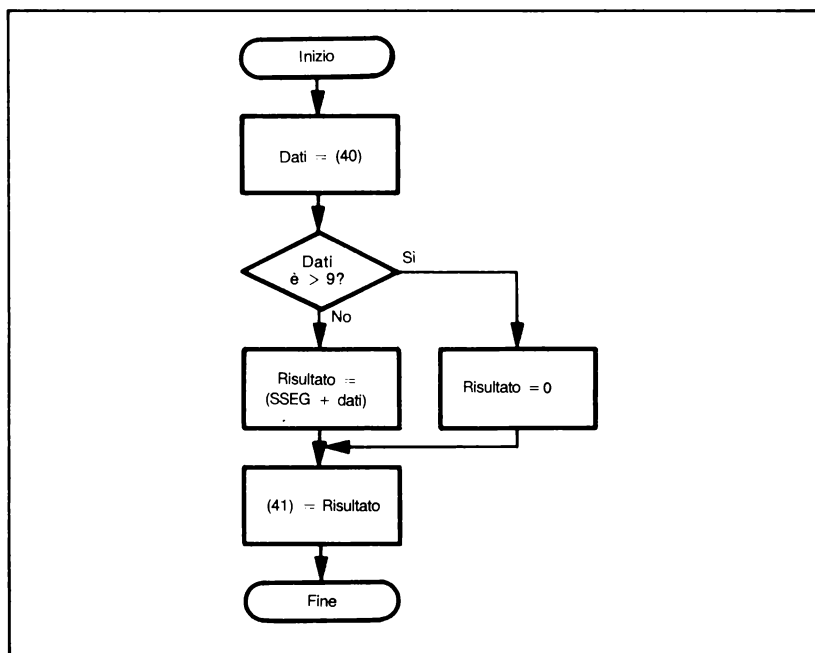



Figura 14-5
Diagramma Di Flusso Della Conversione Da Decimale A 7 Segmenti

1) LHLD SSEG posiziona 3F (esadec.) in L,06 (esadec.) in H.

Questo è chiaramente errato. Si vuole LXI, non LHLD, cioè si vuole l'indirizzo SSEG e non i contenuti di quell'indirizzo.

2) MOV D,A posiziona 0 in D.

Questo è errato — i dati dovrebbero andare in E, poichè si vuole sommarli ai bit meno significativi dell'indirizzo della tabella. Infatti un istruzione dovrebbe azzerare D, poichè l'altra metà della coppia di registri D non è stata inizializzata.

Terzo Programma

```
        MVI    B,0      ;BLANK DISPLAY
        LDA    40H      ;ACCETTA DATI
        CPI    10
        JNC    DONE     ;ATTRAVERSA SE DATI >9
        LXI    H,SSEG   ;ACCETTA INDIRIZZO DELLA TABELLA A 7 SEGMENTI
        MOV    E,A
        MVI    D,0      ;IMPIEGA DATI COME UN INDICE A 16 BIT
        DAD    D        ;TROVA L'INGRESSO DELLA TABELLA MEDIANTE
                        ;INDICIZZAZIONE
        MOV    A,M      ;ACCETTA IL CODICE A 7 SEGMENTI
DONE:   STA    41H
HERE:   JMP    HERE
SSEG:   DB     3FH, 06H, 5BH, 4FH, 69H
        DB     6DH, 7DH, 07H, 7FH, 6FH
```

Il programma produce il seguente risultato:

DATI	RISULTATI
0	3F
9	6F
10	10
6B	6B

Il programma non azzererebbe il risultato se i dati non fossero validi, cioè maggiori di 10. Il programma non trasferisce mai il codice blank nel Registro B all'Accumulatore. Perciò si è alterato il programma sostituendo l'istruzione MOV A,M con MOV B,M. Così, a DONE, il Registro B contiene: a) il codice a 7-segmenti appropriato se nella locazione 40 è presente un numero tra 0 e 9, oppure b) 0 se nella locazione 40 è presente un numero diverso da quelli tra 0 e 9. Per questa ragione un'istruzione MOV A,B è compresa a DONE per muovere i dati corretti nell'Accumulatore quando l'istruzione STA 41 H viene eseguita. Poichè il programma è semplice potrebbe essere convalidato per tutti i digit decimali. I risultati sono:

DATI	RISULTATI
0	3F
1	06
2	5B
3	4F
4	69
5	6D
6	7D
7	07
8	7D
9	6F

Si noti che il numero 8 è errato — esso dovrebbe essere 7F. Poichè tutti gli altri sono corretti, l'errore è quasi sicuramente nella tabella. Infatti l'ingresso 8 è stato mal copiato nella tabella.

Il programma finale è

CONVERSIONE DAL DECIMALE AL CODICE A 7 SEGMENTI

```
        MVI     B,0       ;BLANK DISPLAY
        LDA     40H       ;ACCETTA DATI
        CPI     10
        JNC     DONE      ;ATTRAVERSA SE DATI >9
        LXI     H,SSEG    ;ACCETTA INDIRIZZO DELLA TABELLA A 7 SEGMENTI
        MOV     E,A
        MVI     D,0       ;IMPIEGA DATI COME UN INDICE A 16 BIT
        DAD     D         ;TROVA L'INGRESSO DELLA TABELLA MEDIANTE
                          ;INDICIZZAZIONE
        MOV     B,M       ;ACCETTA IL CODICE A 7 SEGMENTI
DONE:   MOV     A,B
        STA     41H
        HLT
SSEG:   DB      3FH, 06H, 5BH, 4FH, 69H
        DB      6DH, 7DH, 07H, 7FH, 6FH
```

Gli errori incontrati in questo programma sono tipici di quelli che un programmatore in linguaggio assembly dell'Intel 8080 anticiperebbe.

Questi comprendono:

1) Errore di inizializzazione di variabili.
2) Inversione delle diramazioni condizionali.
3) Diramazione non corretta nel caso di uguaglianza.
4) Confusione tra indirizzamento immediato e diretto, cioè dati ed indirizzi.
5) Dimenticanza di distinzione tra dati ad 8 bit ed indirizzi a 16 bit.
6) Diramazione in posizione errata cosicchè una traiettoria attraverso il programma non è corretta.
7) Liste di numeri copiate non correttamente.

Si noti che le istruzioni chiare come ADD, SUB, JMP, ecc. raramente producono qualsiasi problema.

Classificazione in Ordine Descrescente

Il programma classifica un array di numeri binari ad 8 bit senza Segno in ordine decrescente. L'array comincia nella locazione di memoria 41 e la sua lunghezza è contenuta nella locazione 40.

DEBUGGING DI UN PROGRAMMA DI CLASSIFICAZIONE

Programma Iniziale: (dal diagramma di flusso di Figura 14-6)

```
        MVI     B,0       ;FLAG DI SCAMBIO = 0
        MOV     41H       ;CONTEGGIO = LUNGHEZZA DELL'ARRAY
        MOV     C,A
        LXI     H,42H     ;PUNTA ALL'INIZIO DELL'ARRAY
PASS:   MOV     A,M       ;ACCETTA L'ELEMENTO K-esimo
        INX     H
        CMP     M         ;PARAGONA CON L'ELEMENTO (K + 1)-esimo
        JC      CNT       ;SE IL K-esimo È MAGGIORE NON SCAMBIA
        MOV     M,A       ;SCAMBIA SE È MAGGIORE IL (K + 1)-esimo
        INX     H
```

```
CNT:    DCR   C      ;IL PASSAGGIO È COMPLETO?
        JNZ   PASS   ;SÌ
        DCR   B      ;IL FLAG DI SCAMBIO È = 0?
        JM    PASS   ;NO, ESEGUI UN ALTRO PASSAGGIO ATTRAVERSO L'ARRAY
DONE:   JMP   DONE
```

Il controllo manuale mostra che tutti i blocchi del diagramma di flusso sono contenuti nel programma e che tutti i registri sono stati inizializzati. La ramificazione condizionale deve essere esaminata attentamente. La ramificazione interna JC CNT deve forzare una ramificazione se il nuovo valore è minore o uguale al vecchio. Si noti che nel caso di uguaglianza non deve aversi uno scambio poichè questo creerebbe un ciclo senza fine con la commutazione di due elementi uguali.

Si provi l'esempio:

(41) = 30
(42) = 37

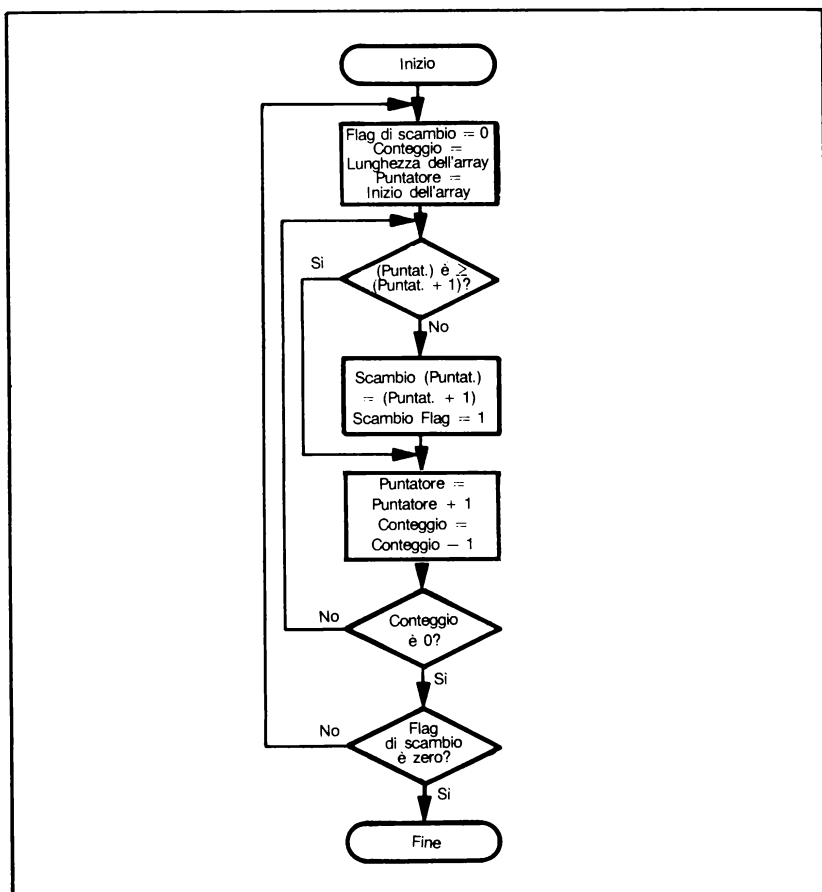



Figura 14-6
Diagramma Di Flusso Di Un Programma Di Classificazione

CMP M si risolve nel calcolo di 30-37. Il Carry è 1. Questo esempio risulterebbe uno scambio ma JNC CNT provvederà alla diramazione corretta in questo caso. Se i due numeri sono uguali il confronto provocherà l'azzeramento del Carry e JNC CNT è ancora corretta.

Perchè JM PASS alla fine del programma? Se non ci sono scambi, B sarà zero cosicchè la diramazione è corretta. Essa dovrebbe essere JP PASS.

Ora si prova l'attraversamento per la prima volta. L'inizializzazione si risolve come segue:

```
   (B)  =  0
   (A)  =  CONTEGGIO
   (C)  =  CONTEGGIO
  (HL)  =  42
```

Gli effetti delle istruzioni del ciclo sono:

```
MOV     A,M      ;(A) = (42)
INX     H        ;(HL) = 43
CMP     M        ;(42) = (43)
JNC     CNT
MOV     M,A      ;(43) = (42)
INX     H        ;(HL) = 44
DCR     C        ;(C) = CONTEGGIO - 1
```

Si noti che le istruzioni di salto condizionato sono già state controllate.

Chiaramente la logica non è corretta. Se i primi due numeri non sono in ordine il risultato dopo la prima iterazione sarebbe:

```
  (42)  =  VECCHIO (43)
  (43)  =  VECCHIO (42)
  (HL)  =  43
   (C)  =  CONTEGGIO - 1
```

Mentre essi sono:

```
  (42)  =  INVARIATO
  (43)  =  VECCHIO (42)
  (HL)  =  44
   (C)  =  CONTEGGIO - 1
```

L'errore in HL è facile da correggere. La seconda istruzione INX H non è necessaria e potrebbe essere omessa. Lo scambio richiede una maggiore attenzione di bit ed un registro temporaneo, cioè:

```
MOV     D,M
MOV     M,A
DCX     H
MOV     M,D
INX     H
```

Uno scambio richiede sempre una locazione di memoria dove un numero può essere immagazzinato mentre l'altro viene trasferito.

Tutti questi cambiamenti richiedono una nuova copia del programma, cioè:

```
        MVI     B,0     ;FLAG DI SCAMBIO = 0
        LDA     41H     ;CONTEGGIO = LUNGHEZZA DELL'ARRAY
        MOV     C,A
        LXI     H,42H   ;PUNTO D'INIZIO DELL'ARRAY
PASS:   MOV     A,M     ;ACCETTA L'ELEMENTO K-esimo
        INX     H
        CMP     M       ;CONFRONTA CON L'ELEMENTO (K + 1)-esimo
        JNC     CNT     ;NON SCAMBIARE SE IL K-esimo È MAGGIORE
        MOV     D,M     ;SCAMBIA GLI ELEMENTI
        MOV     M,A
        DCX     H
        MOV     M,D
        INX     H
CNT:    DCR     C       ;IL PASSAGGIO È COMPLETO?
        JNZ     PASS    ;SÌ
        DCR     B       ;IL FLAG DI SCAMBIO È ZERO?
        JP      PASS    ;NO, ESEGUI UN ALTRO PASSAGGIO ATTRAVERSO
                        ;L'ARRAY
DONE:   JMP     DONE
```

Cosa succede all'ultima iterazione? Diciamo che ci sono tre elementi.

```
(41) = 03
(42) = 02
(43) = 04
(44) = 06
```

Ad ogni attraversamento il programma incrementa HL di 1. Così all'inizio della terza iterazione,

$$(HL) = 42 + 2 = 44$$

Gli effetti dell'istruzione del ciclo sono:

```
MOV     A,M     ;(A) = (44)
INX     H       ;(HL) = (45)
CMP     M       ;(44) = (43)
```

Questo non è corretto; il programma ha tentato di andare oltre la fine dei dati. La precedente iterazione, infatti, dovrebbe essere stata l'ultima perchè il numero di coppie è uguale al numero di elementi meno uno. La correzione consiste nel ridurre di 1 il numero di iterazioni cioè nel posizionare DCR C dopo MOV C,A.

Cosa succede nei casi banali? Cosa succede se l'array non contiene nessun elemento oppure non solo? La risposta è che il programma non lavora correttamente e, senza nessun avvertimento, può cambiare impropriamente un intero blocco di dati (lo si verifichi). Le correzioni per poter manipolare i casi banali sono semplici ma essenziali; il costo è soltanto di poche locazioni di memoria per evitare problemi che potrebbero essere molto difficili da trovare in seguito.

Il nuovo programma è:

```
        MVI   B,0     ;FLAG DI SCAMBIO = 0
        LDA   41H     ;CONTEGGIO = LUNGHEZZA DELL'ARRAY
        CPI   2       ;CONTEGGIO È 2?
        JC    DONE    ;NO, NON È NECESSARIA ALCUNA AZIONE
        MOV   C,A
        DCR   C       ;NO, SE COPPIE = CONTEGGIO - 1
        LXI   H,42H   ;PUNTA ALL'INIZIO DELL'ARRAY
PASS:   MOV   A,M     ;ACCETTA L'ELEMENTO K-esimo
        INX   H
        CMP   M
        JNC   CNT     ;CONFRONTA CON L'ELEMENTO (K + 1)-esimo
        MOV   D,M     ;NESSUNO SCAMBIO SE IL K-esimo È IL MAGGIORE
        MOV   M,A     ;SCAMBIO DEGLI ELEMENTI
        DCX   H
        MOV   M,D
        INX   H
CNT:    DCR   C       ;IL PASSAGGIO È COMPLETO?
        JNZ   PASS    ;SÌ
        DCR   B       ;IL FLAG DI SCAMBIO È 0?
        JP    PASS    ;NO, ESEGUI UN ALTRO PASSAGGIO ATTRAVERSO L'ARRAY
DONE:   JMP   DONE
```

Ora è la volta di controllare il programma sul computer o sul simulatore. In semplice set di dati è:

(41) = 02
(42) = 00
(43) = 01

Questo set consiste di due elementi in ordine errato. Il programma dovrebbe eseguire due passi. Il primo passo raggiungerebbe gli elementi producendo

(42) = 01
(43) = 00
(B) = 01

Il secondo passo completerebbe l'operazione e produrrebbe

(B) = 00

Questo programma è abbastanza lungo per il debugging a fase singola per cui al suo posto si useranno i punti di arresto. Ogni punto di arresto ferma il computer e stampa i contenuti di tutti i registri. I punti di arresto verranno:

1) Dopo LXI H,42H per controllare le condizioni iniziali.
2) Dopo CMP M per controllare il confronto.
3) Dopo la seconda INX H (cioè prima della label CNT) per controllare lo scambio.
4) Dopo DCR B per controllare il completamento di un passo attraverso l'array.

I contenuti dei registri dopo il primo punto di arresto sono:

REGISTRO	CONTENUTI
A	2
B	0
C	1
H	0
L	42

Questi sono tutti corretti, così il programma sta calcolando correttamente le condizioni iniziali in questo caso.

I risultati al secondo punto di arresto sono:

REGISTRO	CONTENUTI
A	0
B	0
C	1
H	0
L	43
CARRY	1

Anche questi risultati sono corretti. I risultati al terzo punto di arresto sono:

REGISTRO	CONTENUTI
A	0
B	0
C	1
D	1
H	0
L	43

Il controllo di memoria mostra:

(42) = 01
(43) = 00

I risultati al quarto punto di arresto sono:

REGISTRO	CONTENUTI
A	0
B	0
C	0
D	1
H	0
L	43

Qui il Registro B non è corretto — il suo valore dovrebbe essere 1 per indicare che si è verificato uno scambio. Infatti un'osservazione al programma mostra che nessuna istruzione segue lo scambio ad ogni cambiamento di B. La correzione consiste nel posizionare l'istruzione MVI B,1 dopo JNC CNT.

Ora la procedura è di caricare il Registro B con il valore corretto e successivo. La seconda iterazione del secondo punto di arresto porge:

REGISTRO	CONTENUTI
A	0
B	0
C	0
H	0
L	44
CARRY	1

Chiaramente il programma ha proceduto non correttamente senza la ri-inizializzazione dei registri (particolarmente HL). Il salto condizionato dopo il controllo del flag di scambio trasferirebbe tutto il controllo al ritorno all'inizio del programma e non alla label PASS.

La versione finale del programma è:

```
SORT:   MVI     B,0      ;FLAG DI SCAMBIO = 0
        LDA     41H      ;CONTEGGIO = LUNGHEZZA DELL'ARRAY
        CPI     2        ;CONTEGGIO È 2?
        JC      DONE     ;NO, NON È NECESSARIA ALCUNA AZIONE
        MOV     C,A
        DCR     C        ;NUMERO DI COPPIE = CONTEGGIO - 1
        LXI     H,42H    ;PUNTA ALL'INIZIO DELL'ARRAY
PASS:   MOV     A,M      ;ACCETTA L'ELEMENTO K-esimo
        INX     H
        CMP     M        ;CONFRONTA CON L'ELEMENTO (K + 1)-esimo
        JNC     CNT      ;NESSUNO SCAMBIO SE IL MAGGIORE È IL K-esimo
        MVI     B,1      ;PONE 1 IL FLAG DI SCAMBIO
        MOV     D,M      ;SCAMBIO ELEMENTI
        MOV     M,A
        DCX     H
        MOV     M,D
        INX     H
CNT:    DCR     C        ;IL PASSAGGIO È COMPLETO?
        JNZ     PASS     ;SÌ
        DCR     B        ;IL FLAG DI SCAMBIO È 0?
        JP      SPORT    ;NO, ESEGUI UN ALTRO PASSAGGIO ATTRAVERSO L'ARRAY
DONE:   JMP     DONE
```

Chiaramente non si possono controllare tutti i casi possibili di questa routine. Altri due semplici set di dati per scopi di verifica sono:

1) Due elementi uguali:

(41) = 02
(42) = 00
(43) = 00

2) Due elementi già in ordine decrescente:

(41) = 02
(42) = 01
(43) = 00

INTRODUZIONE AL TESTING

Chiaramente il testing del programma è strettamente legato al debugging del programma stesso. Sicuramente alcuni dei casi di prova saranno gli stessi dei dati di prova impiegati per il debugging, cioè:

TESTING IMPIEGANTE CASI DAL DEBUGGING

1) Casi banali come assenza di dati oppure un elemento singolo.
2) Casi speciali che il programma esclude singolarmente per qualche ragione.
3) Esempi singoli che impiegano parti particolari del programma.

Nel caso del programma della conversione dal codice decimale a quello a 7 segmenti questi casi coprono tutte le situazioni possibili. I dati di testing consistono di:

1) I numeri da 0 a 9
2) Caso frontiera 10
3) Il caso aleatorio 6B

Il programma non distingue nessun altro caso. Qui il debugging ed il testing sono virtualmente la stessa cosa.
Nel caso di classificazione il problema è molto difficile. Il numero di elementi potrebbe spaziare da 0 a 255 ed ogni elemento potrebbe stare dovunque in quel range. Il numero di casi perciò è enorme. Inoltre il programma è moderatamente complesso. Come selezionare i dati di testing che, in qualche modo, daranno un certo grado di confidenza col programma? Qui il testing richiede qualche decisione di progetto. Il problema del testing è particolarmente difficoltoso se il programma dipende da sequenze dati in tempo reale. Come selezionare i dati da tali sequenze e come presentarli al computer in maniera realistica?

STRUMENTI PER IL TESTING

AIUTI AL TESTING

Molti degli strumenti precedentemente ricordati per il debugging sono comodi anche nello stadio di testing. Gli analizzatori logici o di microprocessori possono servire al controllo dell'hardware; i simulatori software possono servire al controllo del software. Anche altri strumenti possono essere di aiuto, cioè:

1) Simulatori I/O che possono simulare una varietà di dispositivi da uno a singolo ingresso e singola uscita.
2) Emulatori in-circuito che permettono di collegare il prototipo ed eseguire il testing da un sistema di sviluppo o pannello di controllo.
3) Simulatori ROM che hanno la flessibilità di una RAM ma il timing della particolare ROM o PROM che sarà usata nel sistema finale.
4) Sistemi funzionanti in tempo reale che possono fornire ingressi o interrupt in tempi specifici (oppure eseguirli aleatoriamente) e segnalare l'occorrenza di uscite. Punti di arresto in tempo reale e tracce possono anche essere consentiti.
5) Emulazioni (su computer microprogrammabili) che possono fornire velocità di esecuzione in tempo reale ed I/O programmabile.
6) Interfacce che permettono ad un altro computer il controllo del sistema e la convalida del programma del microcalcolatore.
7) Testing di programmi che controllano ogni diramazione di un programma per quanto riguarda gli errori di tipo logico.
8) Programmi di generazione di convalida che possono originare dati aleatori od altre distribuzioni.

Esistono dei teoremi di testing formale ma loro applicabilità è normalmente limitata a programmi molto piccoli.

Occorre fare attenzione al fatto che le forniture di testing non cambino le condizioni ambientali rendendo non significativo il testing stesso. Spesso le forniture di testing possono essere buffer, latch oppure ingressi di condizione o segnali d'uscita. Il sistema effettivo può non fare questo e quindi può comportarsi abbastanza diversamente.

Inoltre ulteriore software presente nell'ambiente di testing può impiegare qualche spazio di memoria o parte del sistema di interrupt. Esso può anche fornire il recupero di errori ed altre caratteristiche che non esisteranno nel sistema finale. Un software test deve essere altrettanto realistico di un hardware test poichè gli errori software possono essere altrettanto critici degli errori hardware.

Le emulazioni ed i simulatori, naturalmente, non sono mai precisi. Essi sono normalmente adeguati per il controllo della logica ma possono talvolta aiutare nel testing della realizzazione dell'interfaccia e del timing. D'altra parte i sistemi in tempo reale non forniscono molto di più di un'analisi della logica e possono avere effetto sulla realizzazione di interfacce e sul timing.

SELEZIONE DEI DATI DI TESTING

Solo pochissimi programmi effettivi possono essere controllati in tutti i casi. Il progettista deve scegliere un campionamento che in qualche modo descriva l'intero settore di possibilità.

TESTING STRUTTURATO

Il testing, naturalmente, dovrebbe essere una parte del processo di sviluppo globale. Il progetto top-down e la programmazione strutturata coinvolgono il testing come parte del progetto. Questo viene chiamato testing «strutturata» (vedere E. Yourdon, Techniques of Program Structure and Program Design, Prentice Hall, Henglewood Cliff, N.J. 1976) secondo la quale viene controllato ogni modulo all'interno del programma strutturato. Così il testing, come la programmazione, diverrebbe modulare, strutturato e top-down.

Questo rimanda al problema della selezione dei dati di testing per un modulo. Il progettista deve prima elencare tutti i casi speciali che un programma consente. Essi possono comprendere:

CASI SPECIALI DI TESTING

1) Casi banali
2) Casi di uguaglianza
3) Situazioni particolari

I dati di testing includerebbero tutti questi.

Si deve ora determinare ogni classe di dati il cui statement all'interno del programma può essere distinguibile.

Essi possono comprendere:

FORMAZIONE DELLE CLASSI DI DATI

1) Numeri positivi o negativi
2) Numeri sopra o sotto una soglia particolare
3) Dati che coinvolgono o no una particolare sequenza di caratteri
4) Dati che sono presenti o no in un particolare istante

Se ogni modulo è abbastanza piccolo il numero totale di classi potrebbe essere ancora abbastanza piccolo anche se queste sono moltiplicative, cioè due decisioni a due vie si risolvono in quattro classi di dati.

SELEZIONE DEI DATI DI TEST DALLE CLASSI

Occorre ora distinguere se il programma produce un diverso risultato per ogni ingresso in una classe (come in una tabella) o produce lo stesso risultato per ogni ingresso nella classe (per esempio in un avvertimento che un parametro è sopra una soglia). Nel caso discreto si può includere ogni elemento se il numero è piccolo o campionarlo se il numero di elementi è grande. Il campione dovrebbe comprendere tutti i casi di frontiera ed almeno un caso selezionato casualmente. Si noti che su alcuni testi sono disponibili tabelle di numeri casuali ed i generatori di numeri casuali fanno parte della possibilità della maggior parte dei computer.

Il progettista deve fare attenzione alle distinzioni che possono non sembrare ovvie. Per esempio, l'Intel 8080 considera un numero ad 8 bit senza segno maggiore di 127 come negativo; il programmmatore deve ricordarsi di questo fatto impiegando le istruzioni di salto JM e JP. Il programmatore deve anche tener conto delle istruzioni che non influenzano i flag, che portano ad overflow per l'aritmetica con segno e delle distinzioni tra la lunghezza di indirizzo (16 bit) e la lunghezza dei dati (8 bit).

ESEMPI DI TESTING

Programma di Classificazione

TESTING DI UN PROGRAMMA DI CLASSIFICAZIONE

I casi speciali sono ovvi:

1) Nessun elemento nell'array
2) Un elemento la cui grandezza può essere selezionata aleatoriamente.

L'altro caso speciale è quello in cui gli elementi sono uguali.

Qui può esserci qualche problema con segno e lunghezza dei dati. Si noti che l'array stesso deve essere lungo meno di 256 elementi. Con l'impiego dell'istruzione MVI B,1 piuttosto che INR B per porre ad '1' il flag di scambio non presenterà difficoltà se il numero di elementi o scambi supera 128. Si potrebbe controllare gli effetti del segno scegliendo metà dei casi di testing da regolare con numeri di elementi tra 128 e 255 e metà tra 2 e 127.

Tutte le grandezze dovrebbero essere scelte aleatoricamente così da evitare il più possibile qualunque predisposizione inconscia.

Numeri Auto-Controllanti

TESTING DI UN PROGRAMMA ARITMETICO

Qui si assumerà che un precedente controllo di validità abbia assicurato che il numero è di lunghezza corretta e consiste di digit validi. Poichè il programma non fa altre distinzioni, i dati di testing potrebbero essere selezionati casualmente. Qui si dimostrerà ideale una tabella di numeri casuali oppure un generatore di numeri casuali; il range dei numeri casuali è da 0 a 9.

PRECAUZIONI SUL TESTING

Il progettista può semplificare la procedura di testing mediante un progetto ragionevole del programma. Si potrebbero impiegare le seguenti regole:

1) Si cerchi di eliminare i casi banali prima possibile senza introdurre distinzioni non strettamente necessarie.
2) Si minimizzi il numero di casi speciali. Ogni caso speciale implica un tempo addizionale di testing e debugging.
3) Si consideri la validità di esecuzione od i controlli di errore sui dati fatti precedentemente all'elaborazione.
4) Si faccia attenzione alle distinzioni involontarie e non necessarie, particolarmente manipolando numeri con segno od impiegando operazioni che fanno riferimento a numeri con segno.
5) Si controllino manualmente i casi frontiera. Essi sono spesso una sorgente di errori. Ci si assicuri che la definizione del problema specifichi cosa succede in questi casi.
6) Si costruisca il programma generale per quanto ragionevolmente possibile. Ogni distinzione e routine separata aumenta il testing richiesto.
7) Si separino e si strutturino i moduli in modo che il testing possa procedere a passi separati, analogamente alle altre fasi del ciclo di sviluppo.

CONCLUSIONI

Il debugging ed il testing sono da considerare i figliastri del processo di sviluppo del software. La maggior parte dei progetti lasciano troppo poco tempo a questi ultimi e la maggior parte dei libri di testo li trascurano. Ma i progettisti ed i dirigenti spesso trovano che questi sono molto lunghi e dispendiosi. L'avanzamento può essere molto difficile da misurare o produrre. Il debugging ed il testing del software del microprocessore è particolarmente difficile a causa degli strumenti limitati disponibili.

Il progettista dovrebbe pianificarli accuratamente. Si raccomanda la seguente procedura:

1) Si cerchi di scrivere dei programmi su cui possa essere facilmente eseguito il debugging ed il testing. La programmazione modulare, quella strutturata ed il progetto top-down sono tutte tecniche convenienti.
2) Si prepari un piano di debugging e testing come parte del progetto del programma. Si decida precedentemente quali dati generare e quali attrezzature si richiederanno.
3) Si esegua il debugging ed il testing di ogni modulo come parte del processo di progetto top-down.
4) Si esegua il debugging sistematico della logica di ogni modulo. Si impieghino liste di controllo, punti di arresto ed il modo a fase singola.
   Se la logica è complessa si consideri il simulatore software.
5) Si controlli sistematicamente il timing di ogni modulo. Un oscilloscopio può risolvere molti problemi. Se si pianifica il test in modo corretto. Se il timing è complesso si consideri un analizzatore logico o di microprocessori.
6) Ci si assicuri che i dati di test siano un campione rappresentativo. Si rilevi per qualunque classe di dati cosa può distinguere il programma.
7) Se il programma manipola diversamente ogni elemento oppure il numero di casi è grande si selezionino aleatoriamente i dati di test.
8) Si registrino tutti i risultati di test come parte della documentazione.

Capitolo 15
DOCUMENTAZIONE E RIPROGETTO

Un programma funzionante che guida un sistema computer non costituisce la fine del ciclo di sviluppo del software. Anche un'adeguata documentazione è una parte importante del prodotto software. Non solo la documentazione aiuterà il progettista nel debugging e testing degli stati ma è anche essenziale per impieghi successivi e per l'estensione del programma. Un programma scarsamente documentato sarà difficoltoso da mantenere, impiegare, estendere.

Occasionalmente la prima versione di un programma non ha essenziali richieste di tempo di esecuzione od impiego di memoria. Il progettista deve poi considerare i modi per migliorare il programma. Questo stadio è chiamato riprogetto e richiede concentrazione in quelle parti del programma che possono produrre la maggior parte dei miglioramenti.

PROGRAMMI AUTO-DOCUMENTANTI

Sebbene nessun programma è sempre completamente autodocumentante alcune delle regole precedentemente ricordate possono essere di aiuto. Queste comprendono:

REGOLE PER I PROGRAMMI AUTO-DOCUMENTANTI

1) Le strutture chiare e semplici si realizzano con il minor numero di trasferimenti di controllo (cioè salti) possibile.
2) Si usino nomi e label significativi.
3) Si usino dei nomi per i dispositivi di I/O, parametri, fattori numerici, ecc.
4) Si sottolinei la semplicità piuttosto che il risparmio di memoria, tempo di esecuzione oppure di rappresentazione.

Per esempio, il semplice programma seguente, invia una stringa di caratteri ad una telescrivente:

```
        LDA     2000H
        LXI     H,1000H
        MOV     B,A
W:      MOV     A,M
        OUT     6
        CALL    XXX
        DCR     B
        INX     H
        JNZ     W
        HLT
```

Anche senza l'aggiunta di commenti, si può migliorare il programma come segue:

```
MESSG   EQU   1000H
COUNT   EQU   2000H
TTY     EQU   6
        LDA   COUNT
        MOV   B,A
        LXI   H,MESSG
OUTCH:  MOV   A,M
        OUT   TTY
        CALL  BITD
        INX   H
        DCR   B
        JNZ   OUTCH
        HLT
```

Chiaramente questo programma è più comprensibile della versione precedente. Anche senza aggiungere ulteriore documentazione, si potrebbe probabilmente indovinare la funzione del programma ed il significato della maggior parte delle variabili. Altre tecniche di documentazione non sono il sostituto dell'auto-documentazione.

Qualche nota ulteriore sulla scelta dei nomi:

SCELTA CONVENIENTE DEI NOMI

1) Impiegare dei nomi ovvi quando sono disponibili; per esempio, TTY o CRT per dispositivi di uscita, START o RESET per indirizzi, DELAY o SORT per subroutine, COUNT o LENGTH per dati.
2) Evitare acronismi come S16BA per SORT 16 BIT ARRAY. Questo tipo di label raramente significa qualcosa per chiunque diverso dal creatore.
3) Si impieghino, dove possibile, parole piene o parti chiuse di parole piene, per esempio, DONE, PRINT, SEND.

COMMENTI

La forma più ovvia di documentazione addizionale è il commento. Comunque pochissimi programmi (comprendendo la maggior parte di quelli dei libri) hanno commenti realmente efficienti. Si dovrebbero considerare le seguenti linee guida per buoni commenti.

LINEE DI GUIDA PER I COMMENTI

1) Non si ripeta il significato del codice d'istruzione. Si spieghi invece qual è lo scopo dell'istruzione nel programma.

   I commenti come

   ```
   DCR     B          ;B = B - 1
   ```

   non aggiungono nulla alla documentazione. Invece si aggiungano commenti come:

   ```
   DCR     B          ;NUMERO LINEA = NUMERO LINEA - 1
   ```

   Si ricordi che si conosce cosa significa il codice di istruzione e che qualunque altro utente può consultare sul manuale i loro significati. Lo scopo essenziale della documentazione è di spiegare cosa sta facendo il programma.

2) Si facciano i commenti più chiari possibile. Non si impieghino abbreviazioni o stenografia a meno che essi non siano veramente ovvi.

Evitare commenti come:

```
DCR     B          ;LN = LN - 1
```

oppure

```
DCR     B          ;DEC.LN BY1
```

la rappresentazione ulteriore è semplicemente tutt'altro che dispendiosa.

3) Si commentino sempre i punti importanti od oscuri. Si faccia particolare attenzione a contrassegnare le operazioni che possono non essere ovvie, come

```
ANI     11011111   ;BIT OFF DEL LETTORE DI NASTRO
```

oppure

```
DAD     D          ;INDICE DELLA TABELLA DEL CODICE GRAY
```

Chiaramente spesso le operazioni di I/O richiedono commenti estensivi. Se non si è esattamente sicuri di cosa fa un'istruzione oppure è necessario pensarci, si scriva all'esterno cosa la funzione esegue. Il commento rimanderà ad un tempo successivo e questo sarà comodo nella documentazione.

4) Non si commentino le cose ovvie. Un commento in ogni riga rende semplicemente difficoltosa la ricerca dei punti importanti. Sequenze standard come:

```
INX     H
DCR     B
JNZ     SRCH
```

non richiedono alcun contrassegno se lo scopo della sequenza è ovvio in virtù della documentazione precedente, per esempio se questa sequenza compare in una subroutine che somma una serie di byte senza segno. Un solo commento spesso sarà sufficiente per diverse linee. Per esempio:

```
RRC                ;SCAMBIO DI DIGIT
RRC
RRC
RRC
MOV     A,C        ;SCAMBIA I BYTE PIÙ E MENO SIGNIFICATIVI
MOV     C,B
MOV     B,A
```

5) Posizionare i commenti nella linea a cui ci si riferisce ovvero all'inizio della sequenza.
6) Si mantengano aggiornati i commenti. Se si cambia il programma, si cambino i commenti. I commenti fuori posto od errati sono peggio che se non ci fossero del tutto.
7) Nei commenti si impieghino termini e forme standard. Non si sbagli su pseudo-ripetizioni, nomi variati per la stessa idea creano confusione, per esempio CONTEGGIO e CONTATORE, PARTENZA ed INIZIO, DISPLAY e LED, PANNELLO ed INTERRUTTORI. Non c'è un guadagno effettivo in queste inconsistenze.
   Le variazioni possono sembrare ovvie sul momento ma non in seguito; altri possono rimanere confusi.
8) Si cerchi di migliorare i commenti. Se si trova un commento che non si riesce a leggere o capire, si trovi il tempo per cambiarlo. Se si trova che una lista è molto addensata si aggiunga qualche riga bianca. I commenti non miglioreranno sè stessi; infatti essi sembreranno accettare il compito indietro e dimenticano esattamente cosa si fa.

L'ultima cosa che va ricordata è che i commenti sono importanti. Se ben fatti risparmieranno tempo e sforzi. Si dedichi quindi del lavoro ad essi per renderli più efficienti possibile.

ESEMPI DI COMMENTO

Addizione a Precisione Multipla

Il programma di base è:

ESEMPI DI COMMENTI

```
        LDA    40H
        MOV    B,A
        LXI    H,41H
        LXI    D,61H
        ANA    A
ADDW:   LDAX   D
        ADC    M
        MOV    M,A
        INX    D
        DCR    B
        JNZ    ADDW
HERE:   JMP    HERE
```

Primo, si commentino i punti importanti. Questi sono tipicamente inizializzazioni, acquisizioni di dati ed operazioni di inizializzazione. Non ci si preoccupi con le sequenze standard come l'aggiornamento di puntatori e contatori se si chiarisce quali registri incrementati/decrementati sono puntati-contati. Si ricordi che i nomi sono più chiari dei numeri così si usino liberamente.

La nuova versione del programma è:

```
LENG    EQU    40H
NUMB1   EQU    41H
NUMB2   EQU    61H
        LDA    LENG     ;CONTEGGIO = LUNGHEZZA DEI NUMERI
        MOV    B,A
        LXI    H,NUMB1 ;INIZIO AGLI LSB DEL PRIMO NUMERO
        LXI    D,NUMB2 ;INIZIO AGLI LSB DEL SECONDO NUMERO
        ANA    A
ADDW:   LDAW   D        ;ACCETTA GLI 8 BIT DEL SECONDO NUMERO
        ADC    M        ;E GLI 8 BIT DEL PRIMO NUMERO
        MOV    M,A      ;IMMAGAZZINA I RISULTATI NEL PRIMO NUMERO
        INX    D
        INX    H
        DCR    B
        JNZ    ADDW
HERE:   JMP    HERE
```

DOMANDE PER I COMMENTI

Secondo, si osservi il programma ricercando qualsiasi istruzione che potrebbe non avere una funzione ovvia e si mettano dei contrassegni. Qui lo scopo di ANA A è di azzerare il Carry. Questo scopo certamente non è evidente dall'istruzione.

Terzo ci si chieda se i commenti chiariscono quanto necessario conoscere se si vuole impiegare il programma, per esempio:

1) Quali parametri sono necessari e quali sono i loro valori iniziali?
2) Quali operazioni esegue il programma?
3) Dove esso accetta i dati?
4) Dove esso immagazzina i risultati?
5) Quali casi speciali esso considera?
6) Cosa fa il programma sugli errori?
7) Come esso esiste?

Alcune delle domande possono non essere rilevanti per un particolare programma ed alcune delle risposte possono essere ovvie. Ci si assicuri che non si debba esaminare accuratamente il programma per trovare la risposta. Non c'è nulla che si potrebbe aggiungere o sottrarre a questa lista? Se no si proceda; si avrà così trovato che il commento è adeguato e ragionevole.

```
LENG    EQU   40H      ;LUNGHEZZA DEI NUMERI
NUMB1   EQU   41H      ;LSB DEL PRIMO NUMERO E RISULTATO
NUMB2   EQU   61H      ;LSB DEL SECONDO NUMERO
        LDA   LENG     ;CONTEGGIO = LUNGHEZZA DEI NUMERI
        MOV   B,A
        LXI   H,NUMB1  ;INIZIO AGLI LSB DEL PRIMO NUMERO
        LXI   D,NUMB2  ;INIZIO AGLI LSB DEL SECONDO NUMERO
        ANA   A        ;CARRY = 0 ALL'INIZIO
ADDW:   LDAX  D        ;ACCETTA GLI 8 BIT DEL SECONDO NUMERO
        ADC   M        ;SOMMA GLI 8 BIT DEL PRIMO NUMERO
        MOV   M,A      ;IMMAGAZZINA IL RISULTATO NEL PRIMO NUMERO
        INX   D
        INX   H
        DCR   B
        JNZ   ADDW
HERE:   JMP   HERE
```

Uscita della Telescrivente

Il programma di base è:

```
        LDA   40H
        ANA   A
        RAL
        MVI   D,11
TBIT:   OUT   TPORT
        RAR
        STC
        CALL  BITD
        DCR   B
        JNZ   TBIT
HERE:   JMP   HERE
```

Commentando i punti importanti ed aggiungendo nomi si ottiene:

```
NBITS:  EQU   11       ;NUMERO DI BIT PER CARATTERE
TDATA:  EQU   40H      ;CARATTERE DA TRASMETTERE
        LDA   TDATA    ;ACCETTA UN CARATTERE
        ANA   A        ;BIT D'INIZIO = 0
        RAL
        MVI   D,NBITS  ;CONTEGGIO = NUMERO DI BIT NEL CARATTERE
TBIT:   OUT   TPORT    ;BIT ALLA TTY
        RAR            ;USCITA SERIALE = BIT SUCCESSIVO
        STC            ;BIT DI ARRESTO = 1
        CALL  BITD     ;ATTENDE 1 TEMPO DI BIT
        DCR   D
        JNZ   TNIT
HERE:   JMP   HERE
```

Si noti come sarebbe semplice cambiare questo programma cosicchè esso trasferisca un'intera stringa di dati, iniziante alla locazione di indirizzo DPTR e DPTR + 1 e terminante con un carattere «04». Inoltre si faccia il dispositivo terminale a 30 caratteri al secondo con un bit di arresto (si dovrà cambiare BITD). Si provi ad eseguire lo scambio prima di osservare la lista.

```
ENDCH   EQU   4        ;CARATTERE FINALE
NBITS   EQU   40       ;NUMERO DI BIT PER CARATTERE
DPTR    EQU   40H      ;(DPTR) = INDIRIZZO DI PARTENZA
        LHLD  DPTR     ;ACCETTA L'INDIRIZZO DI INIZIO DEL MESSAGGIO
TCHAR:  MOV   A,M      ;ACCETTA UN CARATTERE
        CPI   NDCH     ;È IL CARATTERE FINALE?
        JZ    DONE     ;SÌ, DONE
        ANA   A        ;BIT D'INIZIO = 0
        RAL
        MVI   D,NBITS  ;CONTEGGIO = NUMERO DI BIT NEL CARATTERE
TBIT:   OUT   TPORT    ;BIT AL TERMINALE
        RAR            ;USCITA SERIALE = BIT SUCCESSIVO
        STC            ;BIT DI ARRESTO = 1
        CALL  BITD     ;ATTENDE 1 TEMPO DI BIT
        DCR   D
        JNZ   TBIT
        INX   H
        JMP   TCHAR
DONE:   JMP   DONE
```

Buoni commenti possono semplificare la variazione di un programma per ottenere nuove specifiche. Per esempio si provi a cambiare l'ultimo programma cosicchè esso:

1) Inizi ogni messaggio con ASCII STX seguito da un codice di identificazione a 3 digit immagazzinato nelle locazioni di memoria da 30 a 32.
2) Non aggiunga i bit d'inizio ed arresto.
3) Attenda 1 ms tra i bit.
4) Trasmetta 40 caratteri inizianti con l'indirizzo in DPTR e DPTR + 1.
5) Termini ogni messaggio con due consecutivi ASCII ETX.

DIAGRAMMI DI FLUSSO E DOCUMENTAZIONE

È stato già discusso l'impiego dei diagrammi di flusso come strumento di progetto al Capitolo 13. I diagrammi di flusso sono anche comodi nella documentazione particolarmente se:

CENNI SULL'IMPIEGO DEI DIAGRAMMI DI FLUSSO

1) Essi sono non così dettagliati da risultare illeggibili.
2) I punti di decisione sono chiaramente spiegati e contrassegnati.
3) Il diagramma di flusso comprende tutte le diramazioni.
4) Il diagramma di flusso corrisponde alla lista del programma effettiv

I diagrammi di flusso sono comodi se danno un'immagine globale del programma. Diventano invece scomodi se sono più difficoltosi da leggere di una lista ordinaria.

PROGRAMMI STRUTTURATI E DOCUMENTAZIONE

I programmi strutturati (Vedere Capitolo 13) possono anche servire come parte di documentazione se:

1) Nel commento si descrive lo scopo di ogni sezione.
2) Si rende chiaro il settore di ogni condizionale o struttura a ciclo mediante identificazione e contrassegni terminali.
3) Si rende la struttura globale più semplice possibile.
4) Si impiega un linguaggio consistente e ben definito.

Il programma strutturato può aiutare nel controllo della logica o nel suo miglioramento. Inoltre, poichè il programma strutturato è indipendente dalla macchina, esso realizzerà facilmente lo stesso compito su un altro computer.

MAPPE DI MEMORIA

Una mappa di memoria è semplicemente una lista di tutte le assegnazioni di memoria di un programma. La mappa consente di determinare l'importo di memoria richiesta, le locazioni dei dati o subroutine e le parti di memoria non allocata. La mappa è un riferimento manipolabile per la ricerca delle locazioni immagazzinate e punti di ingresso e per la suddivisione della memoria tra diverse routine o programmatori. La mappa fornirà anche facile accesso ai dati e subroutine se vengono richiesti in estensioni successive o nella manutenzione.

Una mappa tipica potrebbe essere:

TIPICA MAPPA DI MEMORIA

Memoria di Programma

Indirizzo	Routine	Scopo
0-2	RESET	TRASFERISCE IL CONTROLLO AL PROGRAMMA PRINCIPALE NELLA LOCAZIONE 40 ESADEC.
38-3A	INTRPT	TRASFERISCE IL CONTROLLO AL SERVIZIO DI INTERRUPT NELLA LOCAZIONE 300 ESADEC.
40-265	MAIN	PROGRAMMA PRINCIPALE
270-280	DELAY	PROGRAMMA DI RITARDO
280-290	DSPLY	PROGRAMMA DI CONTROLLO DEL DISPLAY
300-340	KEYIN	PROGRAMMA DI CONTROLLO INTERRUPT PER LA TASTIERA
		Memoria Dati
1000	NKEYS	NUMERO DI TASTI
1001-1002	KPTR	PUNTATORE DEL BUFFER DELLA TASTIERA
1003-1041	KBFR	BUFFER DELLA TASTIERA
1042-1051	DBFR	BUFFER DEL DISPLAY
1052-105F	TEMP	IMMAGAZZINAMENTO TEMPORANEO
10EC-10FF	STACK	STACK DI RAM

La mappa può anche elencare punti d'ingresso addizionali ed includere una descrizione specifica delle parti non impiegate della memoria.

LISTE DI DEFINIZIONE E DEI PARAMETRI

Le liste di definizione dei parametri all'inizio del programma ed in ogni subroutine rendono più semplice la comprensione ed il cambiamento del programma. Possono essere di aiuto le seguenti regole:

REGOLE PER LA LISTA DI DEFINIZIONE

1) Si separino le locazioni RAM, le unità di I/O, la definizione dei parametri e le costanti del sistema di memoria da ogni altra nelle liste.

2) Quando è possibile si dispongano le liste in modo alfabetico con una descrizione di ogni ingresso.

3) Si assegni ad ogni parametro variabile un nome e lo si includa nella lista. Tali parametri possono includere costanti di timing corrispondenti a particolari tasti o funzioni, schemi di controllo o di mascheratura, inizio e fine, caratteri, soglie, ecc.

4) Si renda costante il sistema di memoria in una lista separata. Queste costanti comprenderanno RESET e gli indirizzi di servizio interrupt, l'indirizzo di partenza del programma, le aree RAM, le aree di stack, ecc.

5) Si assegni un nome ad ogni dispositivo di I/O separato anche se due o più di essi possono occupare la stessa porta fisica del sistema attuale. La separazione renderà molto più semplice l'espansione o la riconfigurazione.

Una lista tipica di definizione sarà:

LISTA TIPICA DI DEFINIZIONE

```
;
;COSTANTI DEL SISTEMA DI MEMORIA
;
RESET   EQU   0       ;INDIRIZZO DI RESET
INTRP   EQU   38H     ;INGRESSO INTERRUPT
START   EQU   40H     ;INIZIO DEL PROGRAMMA PRINCIPALE
KEYIN   EQU   300H    ;PROGRAMMA DI INTERRUPT DELLA TASTIERA
RAMST   EQU   1000H   ;INIZIO DELL'IMMAGAZZINAMENTO DEI DATI
STPTR   EQU   1100H   ;INIZIO DELLO STACK
;
UNITÀ I/O
;
DISPLAY EQU   3       ;PORTA D'USCITA DEL DISPLAY
KBDIN   EQU   3       ;PORTA D'INGRESSO DELLA TASTIERA
KBDOT   EQU   3       ;PORTA D'USCITA DELLA TASTIERA
TTY     EQU   1       ;PORTA DATI TTY
TTYST   EQU   0       ;PORTA DI STATO DELLA TTY
;
;MEMORIA RAM
;
        ORG   RAMST
NKEYS   DS    1       ;NUMERO DI TASTI
KPTR    DS    2       PUNTATORE DEL BUFFER DELLA TASTIERA
KBFR    DS    40H     ;BUFFER DELL'INGRESSO DELLA TASTIERA
DBFR    DS    10H     ;BUFFER DEI DATI DEL DISPLAY
TEMP    DS    14H     ;IMMAGAZZINAMENTO TEMPORANEO
;
;PARAMETRI
;
BOUNCE  EQU   2       ;TEMPO DI ELIMINAZIONE RIMBALZO IN MS
GOKEY   EQU   10      ;IDENTIFICAZIONE DEL TASTO 'GO'
MSCNT   EQU   133     ;CONTEGGIO PER IL RITARDO DI 1 MS
OPEN    EQU   0FH     ;SCHEMA PER I TASTI APERTI
TPULS   EQU   1       ;LUNGHEZZA DI IMPULSO PER I DISPLAY IN MS
;
;DEFINIZIONI
;
ALL1    EQU   0FFH    ;SCHEMA TUTTI UNI
STCON   EQU   80H     ;INSERISCE INIZIO CONVERSIONE
```

Naturalmente gli ingressi RAM normalmente non saranno in ordine alfabetico poichè il progettista deve ordinarli in modo da minimizzare il numero di cambi di indirizzo richiesti nel programma.

ROUTINE DI BIBLIOTECA

La documentazione standard delle subroutine permetterà di costruire una biblioteca di programmi utili. L'idea è di rendere questi programmi più facilmente accessibili possibile. Un formato standard potrebbe permettere a chiunque altro di vedere immediatamente cosa esegue il programma. La miglior procedura è di comporre una forma standard e di impiegarla in modo consistente. Si conservino questi programmi in un modo ben organizzato, per esempio in accordo col processore, linguaggio e tipo di programma e si avrà un pratico set di subroutine. Si ricordi, comunque, che senza organizzazione ed opportuna documentazione, impiegando la biblioteca può essere molto più difficile riscrivere il programma di nuovo. Si ricordi che il debugging richiede una dettagliata comprensione di tutti gli effetti di ogni subroutine.

Tra le informazioni che si richiedono una forma standard ci sono:

FORME DI PROGRAMMI DI BIBLIOTECA STANDARD

1) Scopo del programma.
2) Processore impiegato.
3) Linguaggio impiegato.
4) Parametri richiesti e come essi vengono trasferiti alla subroutine.
5) Risultati prodotti e come essi vengono trasferiti al programma principale.
6) Numero di byte di memoria impiegati.
7) Numero di cicli di clock richiesti. Questo numero può essere una media di una figura tipica oppure può variare largamente. Le esecuzioni di tempo reali, naturalmente, dipenderanno dalla velocità di clock del processore.
8) Registri influenzati.
9) Flag influenzati.
10) Un esempio tipico.
11) Manipolazione dell'errore.
12) Casi speciali.
13) Lista del programma documentato.

Se il programma è complesso, la biblioteca standard è probabilmente più comoda se esegue una singola funzione distinta in un modo ragionevolmente generale.

ESEMPI DI BIBLIOTECA

Somma di Dati

Scopo: Il programma SUM8 calcola la somma di un set di numeri binari ad 8 bit senza segno.

Linguaggio: assemblatore Intel 8080.

Condizioni Iniziali: Indirizzo di partenza del set di numeri da sommare nei registri H ed L, lunghezza del set in A.

Condizioni Finali: somma in A.

Requisiti:

Memoria - 9 byte
Tempo - 19+27N cicli di clock dove N è la lunghezza del set.
Registri - A,B,H,L
Influenzati tutti i flag.

Caso Tipico: (tutti i dati in esadecimale):

Inizio:
(HL) = 60
(A) = 03
(60) = 27
(61) = 3E
(62) = 26
Fine:
(A) = 8B

Manipolazione dell'errore: Il programma ignora tutti i riporti. Il bit Carry riflette solo l'ultima operazione. I contenuti di A devono essere 1 o maggiori.

Lista:

```
;
;SOMMA DI DATI AD 8 BIT
;
SUM 8:   MOV    B,A     ;CONTEGGIO = LUNGHEZZA DEL BLOCCO DI DATI
         SUB    A       ;SOMMA = 0
ADD8:    ADD    M       ;SOMMA = SOMMA + DATO
         INX    H
         DCR    B
         JNZ    ADD8
         RET
```

Conversione dal Decimale al Codice a 7 Segmenti

Scopo: Il programma SEVEN converte un numero decimale nel codice del display a 7-segmenti.

Linguaggio: Assemblatore Intel 8080.

Condizioni Iniziali: Dati in A.

Condizioni Finali: Codice a 7 segmenti in A.

Requisiti:

Memoria - 27 byte comprendono la tabella a 7 segmenti SSEG.
Tempo - 78 cicli di clock se il dato è valido.
Registri - A, B, D, E, H, L.
Influenzati tutti flag.

Caso tipico: (dato in esadecimale)

Inizio
(A) = 05
Fine:
(A) = 66

Manipolazione dell'Errore: Il programma ritorna ad 0FFH se il dato non è un digit decimale.

Lista:

```
;
;CONVERSIONE DA DECIMALE A 7 SEGMENTI
;
SEVEN:   MVI    B,0FFH  ;ACCETTA IL CODICE D'ERRORE
         CPI    10      ;IL DATO È DECIMALE?
         JNC    DONE    ;NO, NON È RICHIESTA ALCUNA CONVERSIONE
         LXI    D,SSEG  ;ACCETTA L'INDIRIZZO DI BASE DELLA
                        ;TABELLA A 7 SEGMENTI
         MVI    H,0     ;POSIZIONA IL DATO IN UN INDICE A 16 BIT
         MOV    AL
         DAD    D       ;INDICE DELLA TABELLA
         MOV    B,M     ;ACCETTA IL CODICE A 7 SEGMENTI
DONE:    MOV    A,B
         RET
SSEG:    DB     3FH, 06H, 5BH, 4FH, 66H
         DB     6DH, 7DH, 07H, 7FH, 6FH
```

Somma Decimale

Scopo: Il programma DEC SM somma due numeri decimali a parola multipla.

Linguaggio: Assemblatori Intel 8080.

Condizioni Iniziali: Indirizzo degli LSB di un numero in H ed L, dell'altro numero in D ed E. Lunghezza dei numeri (in parole) in A. Disposizione dei numeri: inizianti con gli LSB.

Condizioni Finali: Il numero della somma sostituisce l'indirizzo di partenza in H ed L.

Requisiti:

Memoria	- 13 byte.
Tempo	- 19+50N dove è il numero di parole.
Registri	- A, B, D, E, H, L

Influenzati tutti i flag — Carry mostra se la somma origina un riporto.

Caso Tipico: (tutti i dati in esadecimale)

Inizio:

```
      (H)  = 60
     (DE)  = 40
      (A)  =  2
     (60)  = 34
     (61)  = 55
     (40)  = 88
     (41)  = 15
```

Fine:

```
     (60)  = 22
     (61)  = 71
    CARRY  =  0
```

Manipolazione dell'Errore: Il programma non esegue alcun controllo per la validità degli interessi decimali. (A) devono essere maggiore o uguale ad 1. Il programma non esegue un controllo per l'overflow.

Lista:

```
;ADDIZIONE DECIMALE A
;PAROLA MULTIPLA
;
DECSM:  MOV     B,A         ;CONSERVA LA LUNGHEZZA
        SUB     A           ;AZZERA IL CARRY ALL'INIZIO
DECAD:  LDAX    D           ;ACCETTA DUE DIGIT DAL SECONDO NUMERO
        ADC     M           ;OPERA L'ADDIZIONE DECIMALE
        DAA
        MOV     M,A         ;IMMAGAZZINA IL RISULTATO COME PRIMO NUMERO
        INX     D
        INX     H
        DCR     B
        JNZ     DECAD
        RET
```

DOCUMENTAZIONE TOTALE

La documentazione completa del software del microprocessore comprenderà tutti o la maggior parte degli elementi che sono stati ricordati. La documentazione totale può coinvolgere:

PACKAGE DI DOCUMENTAZIONE

1) Diagrammi di flusso generali.
2) Una descrizione scritta del programma.
3) Una lista di tutti i parametri e definizioni.
4) Una mappa di memoria.
5) Una lista documentata del programma.
6) Una descrizione del piano di test e dei relativi risultati.

La documentazione può anche coinvolgere:

7) Diagrammi di flusso del programmatore.
8) Programmi strutturati.

La documentazione non è un argomento da considerare con leggerezza o da lasciare alla fine del ciclo di sviluppo del software. Una documentazione appropriata, combinata con una pratica opportuna di programmazione, non é soltanto una parte importante del prodotto finale ma può anche rendere lo sviluppo più semplice, veloce e più produttivo. Il progettista dovrebbe rendere consistente e completa la parte di documentazione di ogni stadio dello sviluppo del software.

RIPROGETTO

Talvolta il progettista deve comprimere la velocità dell'ultimo microsecondo o l'ultimo byte di ulteriore memoria di un programma. Da quando le memorie sono divenute più grandi e meno dispendiose, il problema della memoria è diventato meno serio. Il problema di tempo, naturalmente è importante solo se l'applicazione è critica rispetto a questa variabile; in molte applicazioni il microprocessore deve normalmente consumare la maggior parte del suo tempo in attesa dei dispositivi esterni, e la velocità del programma non è il maggior fattore.

La compressione dell'ultimo bit di esecuzione di un programma è difficile ma non così importante come ogni scrittore importante potrebbe far credere. In primo luogo la pratica è dispendiosa per le ragioni seguenti:

COSTO DEL RIPROGETTO

1) Essa richiede ulteriore tempo del programmatore che è spesso il costo singolo più grosso nello sviluppo del software.

2) Essa può sacrificare strutture e semplicità con un risultante miglioramento del tempo di debugging e testing.

3) Il programma richiede ulteriore documentazione.

4) I programmi risultanti possono essere difficili da estendere, mantenere od impiegare nuovamente.

In secondo luogo il costo più basso e l'esecuzione possono non essere in realtà giustificabile. Il costo più basso e la più alta esecuzione faranno vendere realmente più unità? Oppure si potrebbe fare di meglio con più caratteristiche orientate all'utente? Le sole applicazioni che sembrerebbero giustificare l'ulteriore sforzo sono quelle di volume molto grande, basso costo e basse esecuzioni dove il costo di un ulteriore chip di memoria sarà lontano dal costo dello sviluppo ulteriore del software. Per ulteriori applicazioni si troverà che si sta giocando con giochi costosi per una ragione non molto valida.

MAGGIORE O MINORE RIORGANIZZAZIONE

Comunque, se si deve perseguire lo scopo i seguenti suggerimenti saranno di aiuto. Primo, si determini quanto più adempimento ovvero quanta memoria in meno sono necessari. Se è necessario migliorarli del 25% o meno si può essere in grado di ottenere questo riorganizzando il programma. Se è più del 25% o è stato fatto un errore di progetto di base, nel caso occorre prendere in considerazione cambiamenti drastici in hardware o software, oppure il progettista del sistema ha assegnato un compito ridicolo al progettista software. Si considererà prima la riorganizzazione e, successivamente, i cambiamenti drastici. Si possono trovare diversi esempi anche al Capitolo 5 di 8080 Programmazione per il progetto logico.

RIORGANIZZAZIONE PER IMPIEGARE MINOR MEMORIA

SALVATAGGIO DI MEMORIA

In generale la riorganizzazione di un programma per impiegare minor memoria può anche risolversi in maggior velocità. Comunque questo non è necessariamente il caso e la velocità più elevata è un traguardo meno comune dell'accorciamento dei programmi. I metodi per ottenere programmi più brevi sono:

1) Sostituire i codici ripetitivi allineati con subroutine. Ci si assicuri, comunque, che le istruzioni CALL e RETURN non portano via la maggior parte del guadagno. Si noti che queste sostituzioni normalmente si risolvono in programmi più lenti a causa del tempo impiegato nel trasferimento avanti indietro del controllo.
2) Si impieghino operazioni di registro dove è possibile. Ma si ricordi il costo delle ulteriori inizializzazioni.
3) Si impieghi lo stack quando è possibile. Lo stack è automaticamente aggiornato ogni volta cosicchè non sono necessarie istruzioni esplicite di aggiornamento.
4) Si elimino gli statement di salto. Si provi a riorganizzare il programma oppure ad impiegare salti indiretti (PCHL) od istruzioni RETURN.
5) Si sfrutti il vantaggio di indirizzi che possono essere manipolati come quantità ad 8 bit. Questo per esempio per la pagina zero e gli indirizzi che sono multipli di 100 esadecimale. Per esempio si può tentare di concentrare tutte le tabelle ROM in una sezione di 100_{16} byte di memoria e tutte le variabili RAM in un altro blocco di memoria di 100_{16} byte.
6) Si organizzino i dati e le tabelle in modo da poterli indirizzare senza preoccupazione di riporti, oppure senza qualunque indicizzazione effettiva.
   Questo permetterà ancora di manipolare indirizzi a 16 bit come quantità ad 8 bit. Si vedano le pagine da 5-1 a 5-5 di 8080 Programmazione per il progetto logico per un esempio.
7) Si impieghino le istruzioni a 16 bit per sostituire due operazioni distinte ad 8 bit. Questo può essere particolarmente comodo in risultati di inizializzazione o di immagazzinamento.
8) Si impieghino i residui risultanti dalle parti precedenti del programma.
9) Si sfruttino i vantaggi di istruzioni come INR M,DCRM ed MVI M, che operano sulla memoria oppure posizionando direttamente i risultati nella memoria.
10) Si impieghino le istruzioni RST per raggiungere le subroutine se non si sta impiegandole per interrupt.
11) Si impieghi INR oppure DCR per cambiare il bit 0 oppure per andare da FF (esadec.) a zero.

RIORGANIZZAZIONE PER IMPIEGARE MENO TEMPO

SALVATAGGIO DEL TEMPO DI ESECUZIONE

Benchè alcuni dei metodi che riducono l'impiego di memoria permettano anche un risparmio di tempo, si può avere generalmente soltanto il salvataggio di un importo apprezzabile di tempo dalla concentrazione di cili frequentemente eseguiti. Anche l'eliminazione completa di un'istruzione che è eseguita solo una volta può salvare al massimo pochi microsecondi. Ma un salvataggio in un ciclo che viene eseguito spesso sarà moltiplicato molte volte.

Perciò, se si deve ridurre il tempo di esecuzione, si proceda come segue:

1) Si determino quali istruzioni vengono eseguite più frequentemente. Si può fare questo manualmente oppure impiegando un simulatore software od altri metodi di testing.
2) Si inizi con i cicli eseguiti più frequentemente e si continui tentando di ridurre il numero di cicli finchè si ottiene la riduzione richiesta.
3) Primo, si veda se non c'è nessuna operazione che può essere portata fuori dal ciclo, cioè calcoli ripetitivi, dati che possono essere posizionati in un registro oppure in uno stack, indirizzi che possono essere posizionati in registri, casi speciali od errori che possono essere manipolati esternamente, ecc. Si noti che questo richiederà ulteriore inizializzazione e memoria ma si risparmierà tempo.
4) Si provi ad eliminare gli stantement di salto. Infatti questi impiegano molto tempo.
5) Si sostituiscano le subroutine con codici allineati. Questo permetterà almeno un risparmio di un CALL e di un RETURN. Si noti che questo metodo può aumentare l'importo di memoria necessaria, ma un CALL impiega ben 17 cicli di clock ed un RETURN impiega 10 cicli.
6) Si impieghi lo stack per l'immagazzinamento temporaneo dei dati.
7) Si impieghi qualunque suggerimento ricordato per il rsiparmio di memoria che diminuisce anche il tempo di esecuzione. Questo comprende l'impiego di indirizzi ad 8 bit, istruzioni a 16 bit, ecc..

MAGGIORE RIORGANIZZAZIONE

Se è necessaria una quantità maggiore del 25% di aumento in velocità o di diminuzione dell'impiego di memoria non ci si preoccupi di riorganizzare il codice. In questi casi occorre rivolgersi ad un esperto.

MIGLIORI ALGORITMI

Il cambiamento più ovvio è un algoritmo migliore. Particolarmente se si stanno eseguendo classificazioni, ricerche o calcoli matematici, si può essere in grado di trovare nella letteratura un algoritmo più veloce o più corto. Le biblioteche di algoritmi sono disponibili in alcune riviste e da gruppi professionali. Si vedano per esempio i riferimenti da 1 ad 8 alla fine del capitolo.

Più hardware può, almeno in parte, sostituire del software. I contatori, registri di scorrimento, moltiplicatori hardware ed altri sommatori veloci possono salvare tempo e memoria. I computer, gli UART, tastiere, codificatori e sommatori più lenti possono risparmiare memoria anche se operano più lentamente.

Altri cambiamenti possono essere di aiuto, cioè:

ULTERIORI MAGGIORI CAMBIAMENTI

1) Una CPU con una lunghezza di parola maggiore sarà più veloce se i dati sono abbastanza lunghi. Tali CPU impiegherà minor memoria totale. I processori a 16 bit, per esempio impiegano la memoria in modo più efficiente di quelli ad 8 bit poichè la maggior parte delle istruzioni sono lunghe una parola.
2) Possono esistere delle versioni della CPU che operano a velocità di clock più elevate. Si ricordi, comunque, che non bisogna rendere più veloce la memoria e le porte I/O ma si dovrà aggiustare qualunque ciclo di ritardo.
3) Due CPU possono essere in grado di fare lo stesso lavoro in parallelo o separatamente se si può dividere il lavoro e risolvere il problema delle comunicazioni.
4) Un processore microprogrammato in modo particolare può essere in grado di eseguire lo stesso programma molto più velocemente.
Il costo, comunque, sarà molto più alto anche se si acquista un simulatore off-the-shelf. Tali simulatori sono largamente disponibili per il processore 8080.
5) Si può trovare un compromesso tra tempo e memoria. La consultazione di tabelle e funzioni ROM saranno molto più veloci degli algoritmi, ma essi occupano una notevole quantità di memoria.

DECISIONE DI UN CAMBIAMENTO MAGGIORE

Questo tipo di problema, nel quale è necessario un notevole miglioramento, normalmente risulta da una mancanza di una pianificazione adeguata nello stadio di progetto. La parte della definizione del problema dovrebbe sussistere per determinate quali processori e metodi saranno adeguati per manipolare il problema. Se si sottovaluta il costo sarà alto. Una soluzione economica può risolvere in una spesa non garantita oppure in un tempo di sviluppo dispendioso. Non si provi ad accettare questo, la soluzione migliore è normalmente di eseguire un progetto appropriato e nel contrassegnare un'insufficienza per esperienza. Se sono stati seguiti dei metodi come i diagrammi di flusso, la programmazione modulare, la programmazione strutturata, il progetto top-down ed una documentazione appropriata si sarà in grado di risparmiare una quantità di difficoltà anche se si è costretti a fare un cambiamento maggiore.

BIBLIOGRAFIA

1) Collected Algorithms from ACM, ACM Inc. PO Box 12105, Church Street Station, New York 10249
2) Chen, T.C., «Automatic Computation of Exponentials, Logarithms, Ratio, and Square Roots» IBM Journal of Research and Development, Vol. 16, pp. 380-388, Luglio 1972.
3) H. Schmid, Decimal Computation, Wiley-Interscience, New York, 1974.
4) Knuth, D.E., The Art of Computer Programming, Vol. 1: Fundamental Algorithms, Addison-Wasley, Reading, Mass., 1967.
6) Knuth, D.E., The Art of Computer Programming, Vol. 3: Sorting and Searching, Addison-Wesley, Reading, Mass., 1973.
7) Carnahan, B. ed al., Applied Numerical Methods, Wiley, New York, 1969.
8) Despain, A.M., «Fourier Transform Computers Using CORDIC Iterations, IEEE Transactions on Computers, Ottobre 1974, pp 993-1001.

Capitolo 16
PROGETTI CAMPIONE

PROGETTO # 1: un Cronometro Digitale

PROCEDURA D'INGRESSO DI UN CRONOMETRO

Scopo: Questo progetto consiste di un semplice cronometro. L'operatore fa entrare due digit (minuti e decine di minuti) dalla tastiera di un computer e poi preme il tasto G0. Il sistema conteggia alla rovescia il tempo rimanente su due display LED a 7 segmenti (si veda il Capitolo 11 per una descrizione delle tastiere codificate e dei display LED).

Hardware: Il progetto impiega una porta d'ingresso ed una di uscita, due display a 7-segmenti, una tastiera a 12 tasti, un invertitore 7404 ed un gate AND 7408. I display possono richiedere driver, invertitori e resistori in dipendenza della loro polarità e configurazione.

La Figura 16-1 mostra l'organizzazione dell'hardware. Le linee d'uscita 0, 1 e 2 sono impiegate per esplorare la tastiera. Le linee d'ingresso 0, 1, 2 e 3 sono impiegate per determinare se nessun tasto è stato premuto. Le linee d'uscita 0, 1, 2 e 3 sono impiegate per inviare i digit BCD ai decodificatori/driver a 7 segmenti. La linea d'uscita 4 è impiegata per attivare i display LED (se la linea 4 è ad '1' i display sono accesi). La linea d'uscita 5 è impiegata per selezionare il display di testa e di coda (la linea d'uscita 5 e '1' per il display di testa, '0' per quello di coda).

La linea comune del display di testa è attiva se la linea 4 è '1' e la linea 5 è '1' mentre la linea comune del display di coda è attiva se la linea 4 è '1' e la linea 5 è '0'.

La linea d'uscita 6 controlla il punto decimale sul display di testa. Esso può essere guidato con un invertitore o semplicemente lasciato acceso.

La tastiera è quella di un computer disponibile per 50 c da una sorgente locale. Essa consiste di 12 interruttori-tasti non codificati disposti in quattro righe e tre colonne. Poichè la scrittura della tastiera non coincide con le righe e le colonne osservate, il programma impiega una tabella per identificare i tasti. Le Tabelle 16-1 e 16-2 contengono le connessioni d'ingresso e d'uscita per la tastiera. Il tasto di punto decimale è presente per convenienza dell'operatore e per espansione futura, il programma attuale non impiega tale tasto.

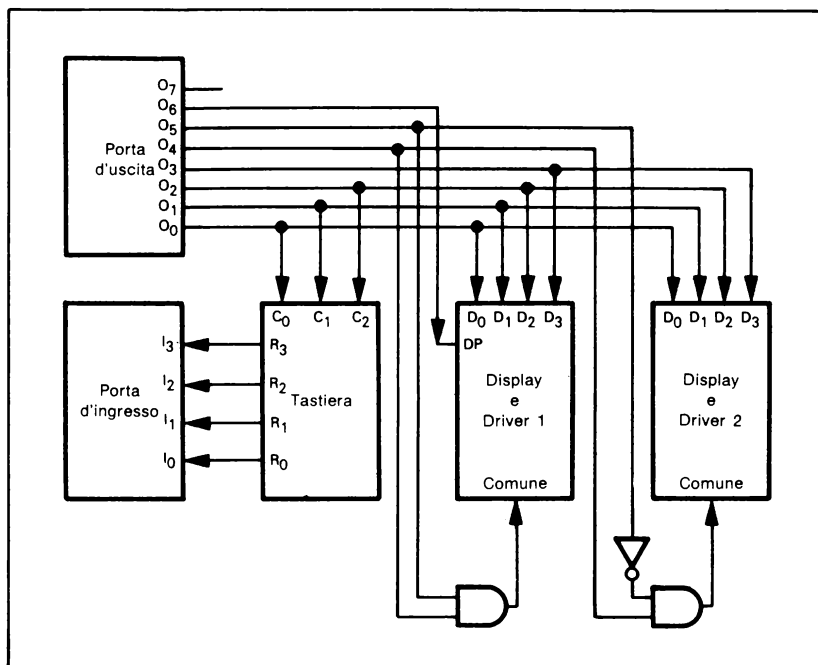



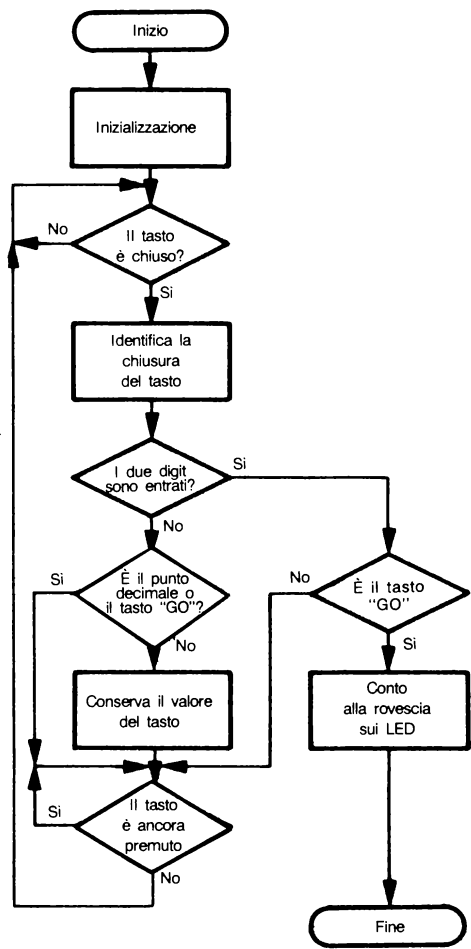
Figura 16-1
Configurazione I/O

Bit d'ingresso	Tasti Connessi
0	'3', '5', '8'
1	'2', '6', '9'
2	'0', '1', '7'
3	'4', '.', 'GO'

Tabella 16-1
Connessioni D'Ingresso Per La Tastiera Del Timer

Bit d'uscita	Tasti Connessi
0	'0', '2', '3', '4'
1	'1', '8', '9', 'GO'
2	'5', '6', '7', '.'

Tabella 16-2
Connessioni D'Uscita Per La Tastiera Del Timer

Diagramma di flusso generale del programma:

Descrizione del Programma:

Il programma è modulare ed ha diverse subroutine. L'enfasi è sulla chiarezza e generalità piuttosto che efficienza; ovviamente il programma non utilizza tutte le capacità del processore 8080. Verranno ora descritte in dettaglio tutte le sezioni della lista.

1) Commenti Introduttori
   I commenti introduttori descrivono pienamente il programma; questi commenti sono un riferimento cosicchè gli altri utenti possano applicare facilmente, estendere e capire il programma. Si noti che i formati standard, gli ordinamenti, e le spaziature aumentano la leggibilità del programma.

2) Definizioni di Variabili
   Tutte le definizioni di variabili sono posizionate all'inizio del programma cosicchè esse possono essere facilmente controllate e cambiate. Ogni variabile è posizionata in ordine alfabetico in una lista con altre variabili dello stesso tipo: i commenti descrivono il significato di ogni variabile. Le categorie sono:

   a) costanti del sistema di memoria, che possono variare da sistema a sistema in dipendenza dello spazio di memoria allocato per programmi diversi o tipi di memorie.
   b) Immagazzinamento temporaneo impiegato per le variabili (RAM).
   c) Numeri dell'unità di I/O.
   d) Definizioni.

   Le costanti del sistema di memoria sono posizionate nelle definizioni cosicchè l'utente può riallocare il programma, immagazzinare temporaneamente ed impilare la memoria senza nessun altro cambiamento. Le costanti di memoria possono essere cambiate per sistemare altri programmi o per farle coincidere con un particolare posizionamento del sistema di indirizzo ROM e RAM.

   L'immagazzinamento temporaneo è allocato per mezzo di una pesudo-operazione DS (Immagazzinamento Definito). Una pseudo-operazione ORG (Origine) posiziona le locazioni di immagazzinamento temporaneo in una zona particolare della memoria. Nessun valore viene posizionato in queste locazioni, cosicchè il programma può essere posizionato su ROM ed il sistema funziona con il ripristino di potenza senza ricaricamento.

   Le unità di I/O sono sempre individuate da un nome cosicchè il numero dell'unità può essere facilmente cambiato per manipolare diverse configurazioni. Ad ogni unità di I/O viene assegnato un nome diverso, sebbene alcune unità sono fisicamente la stessa nella configurazione attuale.

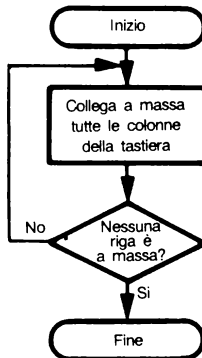
   Le definizioni chiariscono il significato di certe costanti e consentono un facile cambiamento dei parametri. Le definizioni sono date nella forma (per esempio biliaria, esadecimale, ottale, ASCII, decimale) in cui viene chiarito il loro significato. I parametri (come un tempo di eliminazione rimbalzo) vengono qui posizionati cosicchè possano essere variati con le richieste del sistema.

3) Inizializzazione
   La locazione 0 (la locazione di ripristino del microprocessore) contiene un salto al programma principale. Il programma principale, in questo modo, può essere posizionato dovunque nella memoria e raggiunto attraverso il segnale RESET.

   L'inizializzazione consiste di tre fasi:
   a) Posizionamento di un valore di partenza nel Puntatore dello Stack. Lo stack è impiegato solo per immagazzinare gli indirizzi di ritorno della subroutine.
   b) Partenza del numero dei tasti di digit premuti a zero.

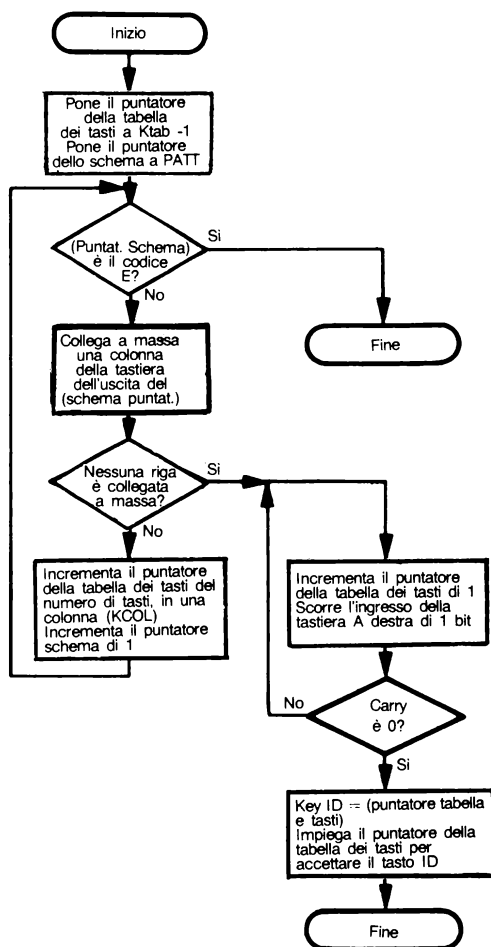
c) Inizializzazione della locazione dove il tasto di digit successivo sarà conservato all'inizio dell'array dei tasti di digit.
La locazione di memoria KEYAD contiene l'indirizzo in cui sarà posizionato il digit successivo. Ogni volta che il programma accetta un tasto di digit esso incrementa i contenuti di KEYAD cosicchè esso posizionerà il tasto di digit successiva nella locazione di memoria successiva.

4) Esame della Chiusura di Tasto

Diagramma di Flusso:

La chiusura di tasto è identificata dal collegamento a massa di tutte le colonne della tastiera e quindi dal controllo delle righe messe a massa (cioè della chiusura dell'interruttore da colonna a riga). Si noti che il programma non assicura alcun valore per i bit d'ingresso non impiegati; invece esso pone a '0' i bit connessi alle righe della tastiera con una istruzione di AND logico.

5) Eliminazione Rimbalzo del Tasto
Il programma elimina il rimbalzo della chiusura del tasto in software attendendo per due millisecondi che si ritiene abbastanza lungo perchè si ottenga un buon contatto. Il numero di millisecondi è contenuto nell'Accumulatore. Il ritardo dovrebbe essere aggiustato se si impiegasse un clock più lento o memorie più lente. Si potrebbe eseguire il cambiamento semplicemente ridefinendo la costante MSCNT.

6) Chiusura del Tasto di Identificazione

Diagramma di Flusso

Il particolare tasto chiuso è identificato dal collegamento a massa delle singole colonne e dall'osservazione se si trova una chiusura. Una volta trovata una chiusura (cosicchè è nota la colonna del tasto), la fila del tasto può essere determinata facendo scorrere l'ingresso.

La struttura richiesta per collegare a massa le singole colonne della tastiera è contenuta nella tabella PATT in memoria. La struttura finale nella tabella è un contrassegno che indica che tutte le colonne sono state collegate a massa senza aver trovato una chiusura. Questa struttura indica anche, al programma principale, che la chiusura potrebbe anche essere non identificata (per esempio la chiusura del tasto sia terminata oppure si sia verificato un errore hardware prima di poter trovare la chiusura).

TABELLA DEI TASTI

Le identificazioni di tasto sono nella tabella KTAB in memoria. I tasti della prima colonna (connesse al bit d'uscita meno significativo) sono seguite da quelle della seconda colonna, ecc. Dentro una colonna il tasto della riga connessa al bit d'ingresso meno significativo è il primo, ecc.. Così ogni volta che una colonna viene esplorata senza trovare una chiusura il numero di tasti in una colonna (KCOL) deve essere sommato all'indirizzo base della tabella per ottenere l'indirizzo di partenza della Colonna successiva. Il puntatore della tabella dei tasti viene anche incrementato di uno prima che ogni bit della riga d'ingresso venga esaminato; questo processo si arresta quando si trova un ingresso zero. Si noti che il puntatore della tabella dei tasti è iniziato una locazione prima della tabella poichè esso è sempre incrementato di uno nella ricerca della riga corretta. Se il programma non può identificare la chiusura di tasto, esso la ignora semplicemente e ricerca un'altra chiusura.

7) Azione di Identificazione di Tasto
   Se il programma ha digit sufficienti (due in questo caso semplice), esso osserva solo il tasto GO ed ignora tutti gli altri. Se esso trova il tasto GO procede alla fase 8 ed inizia il conteggio.

   Se il programma non ha digit sufficienti esso ignora il punto decimale o il tasto GO. Se trova un tasto di digit, esso conserva il valore nell'array dei tasti, incrementa il numero dei tasti premuti ed incrementa il puntatore dell'array dei tasti.

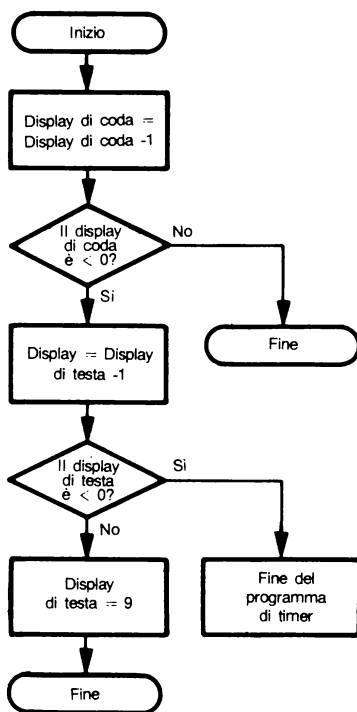
   Se il processo non è completo il programma deve attendere la chiusura di tasto per terminare cosicchè il sistema non leggerà ancora la stessa chiusura. L'utente deve attendere tra chiusure di tasto, cioè arrestare la pressione di un tasto prima di premere un altro.

   Si noti che il programma identificherà le chiusure di tasti doppi, come un tasto o un altro in dipendenza da quale chiusura trova prima la routine di identificazione.

8) Posizionamento d'Uscita del Display.
   I digit sono posizionati nei registri con 4 bit posti ad uno cosicchè l'uscita è inviata al display. I bit 5 e 6 sono posti per il display di testa per guidare correttamente l'uscita e per accendere il punto decimale. L'istruzione di OR logico con una maschera appropriata pone ad uno i bit.

9) Comando dei Display LED.
   Ogni display viene acceso per due millisecondi. Questo programma è ripetuto 1500 volte per ottenere un tempo totale del display di 6 secondi. Gli impulsi sono abbastanza frequenti cosicchè i display LED appaiono essere illuminati continuamente.

10) Conteggio alla Rovescia del Display

Diagramma di Flusso:

Il valore sul display di testa è decrementato di uno. Se questo influenza il bit 4 (LEDON è impiegato per accendere il display), il valore diviene negativo. Deve essere quindi ottenuto un prestito dal display di testa. Se il prestito dal display di testa influenza il bit 4 il conteggio ha raggiunto lo zero ed il conto alla rovescia è terminato.
Diversamente il programma pone a '9' il display di testa e riparte.

Si noti che i commenti descrivono le sezioni del programma ed i singoli statement. I commenti spiegano cosa sta facendo il programma ma non cosa fanno i codici di istruzione specifici.

```
;NOME DEL PROGRAMMA: TIMER
;DATA DEL PROGRAMMA: 4/9/76
;PROGRAMMATORE: LANCE A. LEVENTHAL
;REQUISITI DEL PROGRAMMA: DC (220) PAROLE
;REQUISITI RAM: 5 PAROLE
;REQUISITI I/O: 1 PORTA D'INGRESSO, 1 PORTA D'USCITA
;
;
;QUESTO PROGRAMMA È UN TIMER SOFTWARE CHE ACCETTA INGRESSI
; DA UNA TASTIERA DI COMPUTER E QUINDI ESEGUE UN CONTO ALLA ROVESCIA
; SU DUE DISPLAY LED A 7-SEGMENTI IN MINUTI E DECINE DI MINUTI
;
;TASTIERA
;
;SI ASSUME UNA TASTIERA A 12 TASTI
;LE CONNESSIONI DELLE TRE COLONNE SONO LE USCITE DEL PROCESSORE
; COSICCHÈ UNA COLONNA DI TASTI PUÒ ESSERE COLLEGATA A MASSA
;LE CONNESSIONI DELLE QUATTRO RIGHE SONO INGRESSI DEL PROCESSORE
; COSICCHÈ I CIRCUITI CHIUSI POSSONO ESSERE IDENTIFICATI
;SI ELIMINA IL RIMBALZO DEI TASTI DELLA TASTIERA ATTENDENDO
; DUE MILLESECONDI DOPO CHE È STATA RICONOSCIUTA UNA CHIUSURA DI TASTO
;UNA NUOVA CHIUSURA DI TASTO È IDENTIFICATA ATTENDENDO L'UNO
; DI FINE POICHÈ NON È STATO IMPIEGATO LO STROBE
;LE COLONNE DELLA TASTIERA SONO CONNESSE AI BIT DA 0 A 2
; DEL BUS DI USCITA DEL PROCESSORE
;LE RIGHE DELLA TASTIERA SONO CONNESSE AI BIT DA 0 A 2 DEL
; BUS D'INGRESSO DEL PROCESSORE
;
;DISPLAY
;
;SONO IMPIEGATI DUE DISPLAY LED A 7 SEGMENTI CON DECODIFICATORI SEPARATI
; (7447 O 7448 IN DIPENDENZA DEL TIPO DI DISPLAY)
;GLI INGRESSI DEI DATI DEI CODIFICATORI SONO CONNESSI AI BIT
; DA 0 A 3 DEL BUS DI USCITA DEL PROCESSORE
;IL BIT 4 DEL BUS DI USCITA DEL PROCESSORE È IMPIEGATO PER
; ATTIVARE I DISPLAY LED (IL BIT 4 È AD '1' PER INVIARE DATI AI LED)
;IL BIT 5 DEL BUS DI USCITA DEL PROCESSORE È IMPIEGATO PER
; SELEZIONARE QUALE LED DEVE ESSERE IMPIEGATO (IL BIT 5 È AD
; '1' SE DEVE ESSERE IMPIEGATO IL LED DI TESTA, A '0' SE DEVE
; ESSERE IMPIEGATO QUELLO DI CODA)
;IL BIT 6 DEL BUS DI USCITA DEL PROCESSORE È IMPIEGATO
; PER ILLUMINARE IL LED DEL PUNTO DECIMALE SUL DISPLAY DI TESTA
; (IL BIT 6 È AD '1' SE IL DISPLAY DEVE ESSERE ILLUMINATO)
;
;METODO
;
;FASE 1-INIZIALIZZAZIONE
; LO STACK DI MEMORIA (IMPIEGATO PER GLI INDIRIZZI DI RITORNO
; DELLA SUBROUTINE) È INIZIALIZZATO, IL NUMERO DI TASTI DI
; DIGIT PREMUTE È POSTO A ZERO E L'INDIRIZZO IN CUI SARÀ
; POSIZIONATO IL TASTO DI IDENTIFICAZIONE DEL BIT SUCCESSIVO È
; INIZIALIZZATA AL PRIMO INDIRIZZO DELL'ARRAY DI TASTI DI DIGIT
;FASE 2 – ATTESA PER LA CHIUSURA DI TASTO
; TUTTE LE COLONNE DELLA TASTIERA SONO COLLEGATE A MASSA E LE
; RIGHE DELLA TASTIERA VENGONO ESAMINATE FINCHÈ NON SI TROVA
; UN CIRCUITO CHIUSO
```

```
;FASE 3 — ELIMINAZIONE RIMBALZO DI CHIUSURA DI TASTO
; VIENE INTRODOTTA UN'ATTESA DI 2 MS PER ELIMINARE IL RIMBALZO
; DEL TASTO
;FASE 4 — IDENTIFICAZIONE DELLA CHIUSURA DI TASTO
; LA CHIUSURA DI TASTO È IDENTIFICATA COLLEGANDO A MASSA
; OGNI SINGOLA COLONNA DELLA TASTIERA E DETERMINANDO LA RIGA
; E COLONNA DELLA CHIUSURA DI TASTO
; UNA TABELLA VIENE IMPIEGATA PER CODIFICARE I TASTI
; SECONDO I LORO NUMERI DI RIGA E COLONNA
; NELLA TABELLA DEI TASTI I DIGIT SONO IDENTIFICATI DAI LORO
; VALORI, IL PUNTO DECIMALE È IL TASTO NUMERO 10 ED IL TASTO
; «GO» IL NUMERO 11
;FASE 5 — CONSERVA LA CHIUSURA DI TASTO
; LE CHIUSURE DI TASTO DI DIGIT SONO CONSERVATE NELL'ARRAY
; DEI TASTI DI DIGIT ED IGNORATE FINCHÈ NON SONO STATI IDENTIFICATI DUE
; CIFRE IL PUNTO DECIMALE, ULTERIORI CIFRE E CHIUSURA DEL TASTO «GO»
; DOPO CHE SONO STATE TROVATE DUE CIFRE, IL TASTO «GO» È
; IMPIEGATO PER INIZIARE IL PROCESSO DI CONTO ALLA ROVESCIA
;FASE 6 — INTERVALLO DEI TIMER DEL CONTO ALLA ROVESCIA SUL LED
; VIENE ESEGUITO UN CONTO ALLA ROVESCIA SUI LED CON LA CIFRA
; DI TESTA RAPPRESENTANTE IL NUMERO DI MINUTI RIMANENTE E LA
; CIFRA DI CODA LE DECINE DI MINUTI
;
;
;
;DEFINIZIONI DELLE VARIABILI DEL TIMER
;COSTANTI DEL SISTEMA DI MEMORIA
;
;BEGIN   EQU    50H        ;BEGIN È LA LOCAZIONE DI MEMORIA DI
                           ;PARTENZA DEL PROGRAMMA
LASTM    EQU    1000H      ;LASTM È L'INDIRIZZO DI PARTENZA DELLO STACK
TEMP     EQU    800H       ;TEMP È LA PARTENZA DELL'IMMAGAZZINAMENTO
;RAM TEMPORANEO
         ORG    TEMP
KEYAD:   DS     2          ;KEYAD CONSERVA L'INDIRIZZO NELL'ARRAY DEI
                           ;TASTI DI DIGIT
                           ;IN CUI L'IDENTIFICAZIONE DEL
                           ;TASTO DEL NUMERO SUCCESSIVO SARÀ POSIZIONATO
KEYNO:   DS     2          ;KEYNO È L'ARRAY DI TASTO DI DIGIT. ESSO
                           ;CONSERVA LE IDENTIFICAZIONI DEI TASTI DI
                           ;DIGIT
                           ;CHE SONO STATI PREMUTI
NKEYS:   DS     1          ;NKEYS CONSERVA IL NUMERO DI TASTI DI NUMERO
                           ;PREMUTI
;UNITÀ   I/O
;
KBDIN    EQU    3          ;UNITÀ D'INGRESSO PER LA TASTIERA
KBDOT    EQU    3          ;UNITÀ DI USCITA PER LA TASTIERA
LDOUT    EQU    3          ;UNITÀ DI USCITA PER I DISPLAY LED
;
;DEFINIZIONI
DECPT    EQU    01000000B  ;CODICE PER ACCENDERE IL LED DEL PUNTO
                           ;DECIMALE
ECODE    EQU    0FFH       ;CODICE DI ERRORE PER IL NUMERO DI
                           ;CHIUSURE DI
                           ;TASTO TROVATE
```

```
GOKEY   EQU     11          ;NUMERO DI IDENTIFICAZIONE PER LA CHIAVE
                            ;«GO»
KCOL    EQU     4           NUMERO DI TASTI IN UNA COLONNA
LEDON   EQU     00010000B   ;CODICE DA INVIARE ALL'USCITA DI LED
LEDSL   EQU     00100000B   ;CODICE PER SELEZIONARE IL DISPLAY
                            ;DI TESTA
MSCNT   EQU     131         ;CONTEGGIO RICHIESTO PER DARE
                            ;UN TEMPO DI RITARDO DI 1 MS
MXKEY   EQU     2           ;MASSIMO NUMERO DI CHIUSURE DI TASTI DI
                            ;DIGIT CONSENTITO
OPEN    EQU     00001111B   ;INGRESSO DALLA TASTIERA SE NESSUN
                            ;TASTO È CHIUSO
TPULS   EQU     2           ;NUMERO DI MS TRA I DISPLAY DELLE CIFRE
TWAIT   EQU     2           ;NUMERO DI MS PER ELIMINARE IL RIMBALZO
;
;
;
        ORG     0
;
;ROUTINE DI RIPRISTINO PER RAGGIUNGERE IL PROGRAMMA DEL TIMER
        JMP     BEGIN       ;TROVA IL PROGRAMMA DEL TIMER
;INIZIALIZZAZIONE DEL PROGRAMMA DEL TIMER
;
        ORG     BEGIN
        LXI     SP,LASTM    ;PONE LO STACK ALLA FINE DELLA MEMORIA
        XRA     A
        STA     NKEYS       ;INIZIALIZZA IL NUMERO DEI TASTI
                            ;DI DIGIT PREMUTI A ZERO
        LXI     H,KEYNO     ;LOCAZIONE INIZIALE PER LA
                            ;CONSERVAZIONE DEI TASTI DI DIGIT
        SHLD    KEYAD
;
;ESPLORA LA TASTIERA ALLA RICERCA DI CHIUSURA DI TASTO
;
START   CALL    SCANC       ;ATTENDE CHIUSURA DI TASTO
;
ATTESA PERCHÈ SIA ELIMINATO IL RIMBALZO DEL TASTO
;
        MVI     A,TWAIT     ;ACCETTA IL TEMPO DI ELIMINAZIONE RIMBALZO IN MS
        CALL    DELAY       ;ATTESA PER L'ELIMINAZIONE RIMBALZO DEL TASTO
;
;IDENTIFICAZIONE DEL TASTO CHE È STATO PREMUTO
;
        CALL    IDKEY       ;IDENTIFICA LA CHIUSURA DI TASTO
        CPI     ECODE       ;LA CHIUSURA DI TASTO È STATA IDENTIFICATA
                            ;MEDIANTE L'ESPLORAZIONE DI COLONNA?
        JZ      START       ;SE NO ATTENDE PER UN'ALTRA CHIUSURA
;ESEGUE UNA IDENTIFICAZIONE DI TASTO
        MOV     B,A         ;CONSERVA IL NUMERO DEL TASTO
        LXI     H,NKEYS
        MOV     A,M         ;CONTROLLA IL NUMERO MASSIMO DI TASTI DI DIGIT
        CPI     MKEY
        JZ      KEF         ;SE SI OSSERVA IL TASTO «GO»
        MOV     A,B
                            SE NO IGNORA IL PUNTO DECIMALE OPPURE
                            ;IL TASTO «GO»
```

```
        CPI     10
        JNZ     WAITK
        INR     M           ;INCREMENTA IL NUMERO DI TASTI DI DIGIT PREMUTI
        LHLD    KEYAD       ;CONSERVA L'IDENTIFICAZIONE DEL TASTO DI
                            ;DIGIT PREMUTO
        MOV     M,A
        INX     H           ;MUOVI IL PUNTATORE PER CONSERVARE IL TASTO DI
                            ;DIGIT SUCCESSIVO
        SHLD    XEYAD
;
;ATTENDE IL TERMINE DELLA CHIUSURA DEL TASTO ATTUALE
;
WAITK:  CALL    SCAND       ;ATTENDE PER IL CIRCUITO APERTO
        JMP     START       ;OSSERVA IL TASTO SUCCESSIVO
;
;OSSERVA IL TASTO «GO» SE SONO STATE TROVATE ENTRAMBE LE CIFRE
;
KEYF:   MOV     AB          ;ACCETTA IL NUMERO DI TASTO PREMUTO
        CPI     GOKEY       ;CONTROLLA IL TASTO «GO»
        JNZ     WAITK       ;IGNORA IL TASTO SE DIVERSO DA «GO»
;
;PONE LE CIFRE NEI REGISTRI PER IL DISPLAY
;
        LXI     H,KEYND
        MOV     A,M         ;ACCETTA LA CIFRA DI TESTA
        ORI     DECPT       ;ACCENDI IL PUNTO DECIMALE DELLA CIFRA
                            ;DI TESTA
        ORI     LEDON       ;PONE L'USCITA AI LED
        ORI     LEDSL       ;SELEZIONA IL DISPLAY DI TESTA
        MOV     D,A
        INX     H
        MOV     A,M         ;ACCETTA LA CIFRA SUCCESSIVA
        ORI     LEDON       ;PONE L'USCITA AI LED
        MOV     E,A
;
;PULSA I DISPLAY LED
;
LEDLP:  MVI     H,6         ;PONE I CONTATORI PER 6 SECONDI
TLOOP:  MVI     L,250
LDPUL:  MOV     A,D         ;ACCETTA LA CIFRA DI TESTA
        OUT     LDOUT       ;USCITA AL LED 1
        MVI     A,TPULS     ;RITARDO TRA LE CIFRE
        CALL    DELAY
        MOV     A,E         ;ACCETTA LA CIFRA DI CODA
        OUT     LDOUT       ;USCITA AL LED 2
        MVI     A,TPULS     ;RITARDO TRA LE CIFRE
        CALL    DELAY
        DCR     L           ;CONTO ALLA ROVESCIA DEL REGISTRO L
        JNZ     LDPUL
        DCR     H           ;CONTO ALLA ROVESCIA DEL REGISTRO H
        JNZ     TLOOP
;
;DECREMENTA IL CONTEGGIO SUI DISPLAY LED
;
        DCR     E           ;CONTO ALLA ROVESCIA DELLA CIFRA DI CODA
```

```
        MOV     A,E
        ANI     LEDON
        JNZ     LEDLP       ;VA BENE SE NON È RICHIESTO PRESTITO
        DCR     D           ;SE È RICHIESTO PRESTITO CONTO ALLA
                            ;ROVESCIA DELLA CIFRA DI TESTA
        MOV     A,D
        ANI     LEDON
        JZ      BEGIN       ;ATTRAVERSA SE IL CONTEGGIO ATTRAVERSA LO
                            ;ZERO
        MVI     A,9         ;ALTRIMENTI PONE LA CIFRA DI TESTA A 9
        ORI     LEDON       ;PONI L'USCITA AI LED
        MOV     E,A
        JMP     LEDLP       ;RITORNA ALLA SEZIONE DISPLAY
;
;LA SUBROUTINE SCANC ESPLORA LA TASTIERA ATTENDENDO UNA CHIUSURA DI TASTO
;TUTTI GLI INGRESSI DELLA TASTIERA SONO COLLEGATI A MASSA
SCANC:  XRA     A
        OUT     KBDOT       ;COLLEGA A MASSA TUTTE LE COLONNE DELLA
                            ;TASTIERA
        IN      KBDIN
        ANI     OPEN        ;IGNORA GLI INGRESSI NON IMPIEGATI
        CPI     OPEN        ;CONTROLLA I CIRCUITI CHIUSI
        JZ      SCANC       ;CONTINUA LA SCANSIONE SE NESSUN TASTO
                            ;È CHIUSO
        RET
;
;
;LA SUBROUTINE DELAY ATTENDE IL NUMERO DI MILLISECONDI SPECIFICATO
;NEL REGISTRO A MEDIANTE IL CONTEGGIO CON IL REGISTRO C
DELAY:  MVI     C,MSCNT     ;CARICA IL REGISTRO C PER UN RITARDO
                            ;DI 1 MS
WTLP;   DCR     C           ;ATTENDE 1 MS
        JNZ     WTLP
        DCR     A           ;CONTO ALLA ROVESCIA DEL NUMERO DI MS
        JNZ     DELAY
        RET
;
;
;LA SUBROUTINE IDKEY DETERMINA I NUMERI DI RIGA E COLONNA DELLA
;CHIUSURA DI TASTO ED IDENTIFICA IL TASTO SECONDO UNA TABELLA
TDKEY   LXI     B,PATT      ;PUNTA AGLI SCHEMI DI SCANSIONE PER
                            ;COLLEGARE A MASSA UNA COLONNA
        LXI     H,KTAB-1    ;INIZIA IL PUNTATORE DELLA TABELLA DEI TASTI
        LXI     D,KCOL      ;ACCETTA IL NUMERO DI TASTI IN UNA COLONNA
;
;ESPLORA LE COLONNE DELLA TASTIERA SEQUENZIALMENTE OSSERVANDO
CHIUSURE
;
FCOL:   LDAX    B           ;ACCETTA LO SCHEMA PER COLLEGARE A MASSA
                            ;UNA PARTICOLARE COLONNA
        CPI     ECODE       ;TUTTE LE COLONNE SONO STATE ESPLORATE?
        RZ                  ;RITORNA CON CODICE DI ERRORE SE NON È
                            ;STATA TROVATA CHIUSURA
        OUT     KBDOT       ;ESPLORA COLONNA
        IN      KBDIN
```

```
        ANI     OPEN        ;IGNORA GLI INGRESSI NON IMPIEGATI
        CPI     OPEN        ;CONTROLLA I CIRCUITI CHIUSI
        JNZ     FROW        ;COLONNA DETERMINATA SE TROVATA LA CHIUSURA
        DAD     D           ;MUOVE IL PUNTATORE DELLA TABELLA DEI TASTI
                            ;ALLA COLONNA SUCCESSIVA
        INX     B           ;PUNTA ALLO SCHEMA DI SCANSIONE SUCCESSIVO
        JMP     FCOL
;
;DETERMINA IL NUMERO DELLA RIGA DI CHIUSURA
;
FROW    TNX     H           ;MUOVE IL PUNTATORE DELLA TABELLA DEI TASTI
                            ;ALLA RIGA SUCCESSIVA
        RRC                 ;SCORRE GLI INGRESSI RICERCANDO UNA RIGA A MASSA
        JC      FROW        ;PROSEGUE LO SCORRIMENTO DEGLI INGRESSI
                            ;FINCHÈ NON È STATA TROVATA LA CHIUSURA
;
;IDENTIFICA IL TASTO DALLA TABELLA
;
        MOV     A,M         ;ACCETTA IL NUMERO DI TASTO
        RET
;SCHEMI DI SCANSIONE IMPIEGATI PER COLLEGARE A MASSA
; UNA PARTICOLARE COLONNA
;SCHEMA DI ERRORE IMPIEGATO PER INDICARE CHE TUTTE
; LE COLONNE SONO STATE ESPLORATE
;LA COLONNA CONNESSA AL BIT DI USCITA 0 È ESPLORATA PER
; PRIMA, POI QUELLA CONNESSA AL BIT DI USCITA 1, ECC.
;PATT:  DB  00000110B, 00000101B, 00000011B, ECODE.
;TABELLA DELLA TASTIERA
;LE COLONNE COSTITUISCONO L'INDICE PRIMARIO, LE RIGHE
; QUELLO SECONDARIO
;I TASTI DELLA COLONNA CONNESSA AL BIT D'USCITA 0 SONO
; SEGUITE DA QUELLE DELLA COLONNA CONNESSA AL BIT D'USCITA 1,
; ECC. ALL'INTERNO DI UNA COLONNA IL TASTO CONNESSO AL BIT
; D'INGRESSO È PRIMA, SEGUITA DA QUELLA CONNESSA AL BIT
; D'INGRESSO 1, ECC.
;I TASTI DI CIFRA SONO DA 0 A 9, IL PUNTO DECIMALE È 10
; E «GO» È 11
KTAB  DB  3, 2, 0, 4, 8, 9, 1, 11, 5, 6, 7, 10
;
;
;LA SUBROUTINE SCAND ESPLORA LA TASTIERA ATTENDENDO UNA
; CHIUSURA DI TASTO PER TERMINARE CON LA RICERCA DI UNA CHIUSURA
; SUCCESSIVA
SCAND   XRA     A
        OUT     KBDOT       ;COLLEGA A MASSA TUTTE LE COLONNE DELLA
                            ;TASTIERA
        IN      KBDIN
        ANI     OPEN        ;IGNORA GLI INGRESSI NON IMPIEGATI
        CPI     OPEN        ;CONTROLLA I CIRCUITI CHIUSI
        JNZ     SCAND       ;CONTINUA LA SCANSIONE SE IL TASTO
                            ;NON È STATO TROVATO
        RET
        END
```

PROGETTO # 2: Un Termometro Digitale

Scopo: Questo progetto è un termometro digitale che sente la temperatura mediante un termistore (attraverso un convertitore analogico-digitale) e mostra la temperatura in gradi Celsius su due display a 7-segmenti.

Hardware: Il progetto impiega una porta d'ingresso ed una porta d'uscita, due display a 7-segmenti, un invertitore 7404, un gate NEND 7400 oppure un AND 7408 dipendentemente dalla polarità dei display, un convertitore A/D monolitico ad 8 bit, AD7570J Analog Devices, un comparatore LM311 e vari driver di periferica, resistori e condensatori come richiesti dai display ed il convertitore (vedere il Capitolo 11 per una discussione sui convertitori A/D. Si veda anche Hnatek, E.R., A User's Handbook of D/A and A/D Converters, Wiley, New York, 1976; Finkey, J. Computer-Aided Experimentation, Wiley, New York, 1975; Engineering Shaff of Analog Devices Inc., Analog-Digital Conversion Handbook, Analog Devices Inc., P.O. Box 796, Norwood, MA, 1972).

La Figura 16-2 mostra l'organizzazione dell'hardware. La linea d'uscita 7 è impiegata per inviare un segnale d'INIZIO CONVERSIONE al convertitore A/D. Le linee d'ingresso da 0 a 7 sono connesse direttamente alle otto linee dati digitali provenienti dal convertitore. Le linee di uscita da 0 a 3 sono impiegate per inviare i digit BCD ai codificatori/driver a 7-segmenti. La linea di uscita 4 attiva i display e la linea di uscita 5 seleziona il display di testa o di coda (la linea 5 è '1' per il display di testa).

HARDWARE DEL TERMOMETRO ANALOGICO

La parte analogica dell'hardware è mostrata in Figura 16-3. Il termistore fornisce semplicemente una resistenza che dipende dalla temperatura. La Figura 16-4 è un diagramma della resistenza e la Figura 16-5 mostra il range di corrente nel quale la resistenza è indipendente dalla corrente. La conversione a gradi Celsius nel programma è eseguita con una tabella di calibrazione. I vari potenziometri possono essere aggiustati per una scala opportuna dei dati. Per il convertitore A/D viene generato un clock da una rete RC come mostrato in Figura 16-6. I valori sono R = 33 KΩ, C = 1000 pF cosicchè la frequenza di clock è circa 75 KHz. A questa frequenza il tempo massimo di conversione per otto bit è circa 10 microsecondi. La conversione consente un ritardo molto più lungo cosicchè non è necessario un controllo per il termine di conversione.

La versione ad 8 bit del convertitore richiede le seguenti connessioni speciali. Le otto linee dati sono da DB2 a DB9 (DB1 è sempre al livello alto durante la conversione e DB0 basso). L'ingresso ad 8 bit del Ciclo Corto (pin 26-SC8; non indicato) è mantenuto al livello basso cosicchè viene eseguita solo una conversione ad 8 bit. Nel caso attuale l'Abilitazione di Byte Alto (pin 20-HBEN) e l'Abilitazione di Byte Basso (pin 21-LBEN; non indicato) sono entrambe mantenute alte cosicchè le uscite dati sono sempre abilitate. Un buffer tri-state (la porta d'ingresso è mostrata in Figura 16-1) isola le uscite dal Bus Dati del processore.

Il convertitore impiega il metodo delle approssimazioni successive. Esso controlla ogni bit col comparatore analogico per vedere il bit deve essere '0' od '1'. Ogni confronto impiega un periodo di clock. Il metodo è soggetto ad errore se l'ingresso è rumoroso o cambia durante il periodo di conversione.

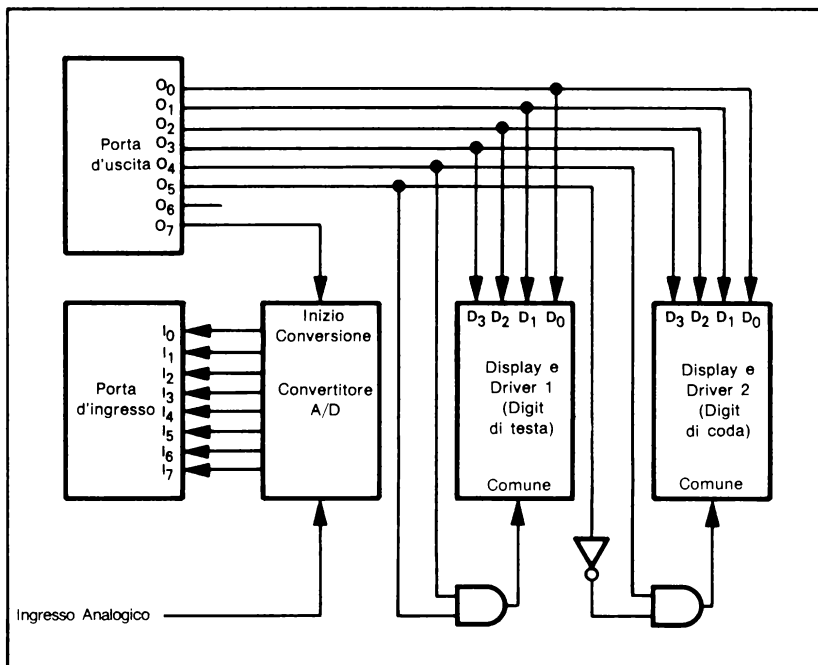
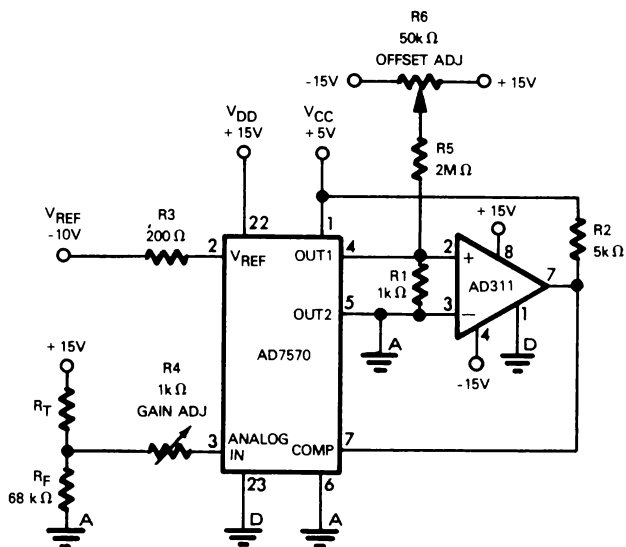



Figura 16-2
Configurazione I/O

Note: Se si impiega V_{REF} positiva l'ingresso Analogico va da 0 a $-V_{REF}$ e l'ingresso (–) del COMPARATORE sarebbe connesso ad OUT 1 (più 4) dell'AD7570.

R_T è il termistore – L'ingresso analogico dal partitore di tensione è:

$$\frac{R_F}{R_F + R_T} \times 15 \text{ Volt}$$

Poichè $R_F = 68\ k\ \Omega$, l'ingresso è

$$\frac{1.02 \cdot 10^6}{R_F + 6.8 \cdot 10^4} \text{ Volt}$$

R_T ha un valore minimo di $3.4 \times 10^4\ \Omega$ (T = 50 °C, Vedere Fig. 16-4) cosicchè il fondo scala è 10 volt.

Figura 16-3
Hardware Analogico

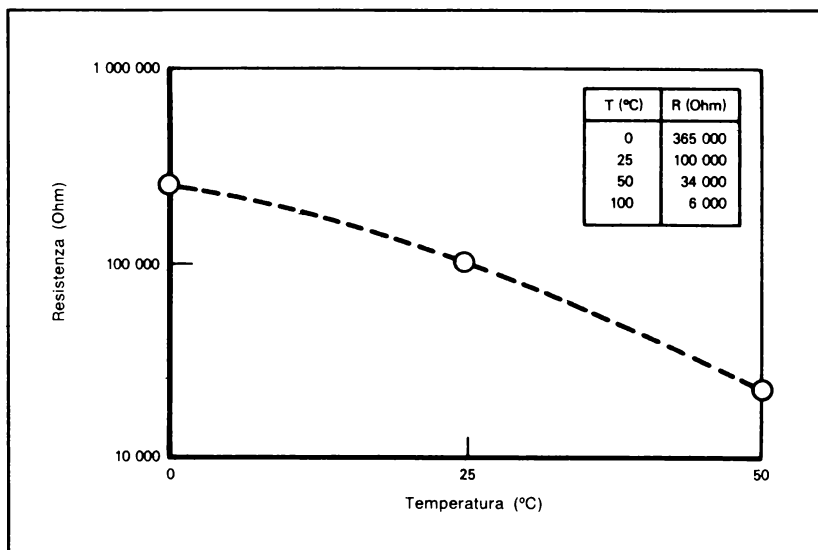



Figura 16-4
Caratteristiche Del Termistore (Fenwal GA51J1 Bead)

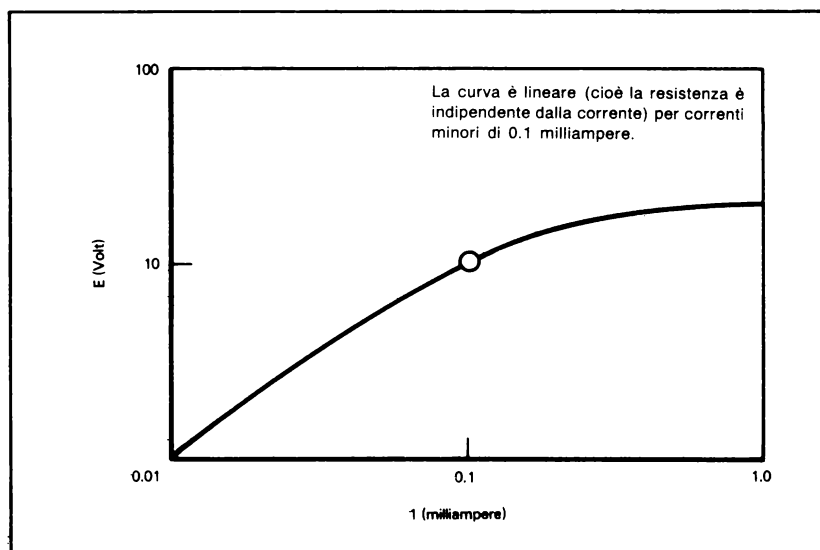



Figura 16-5
Curva E - I Tipica Del Termistore (25 °C)

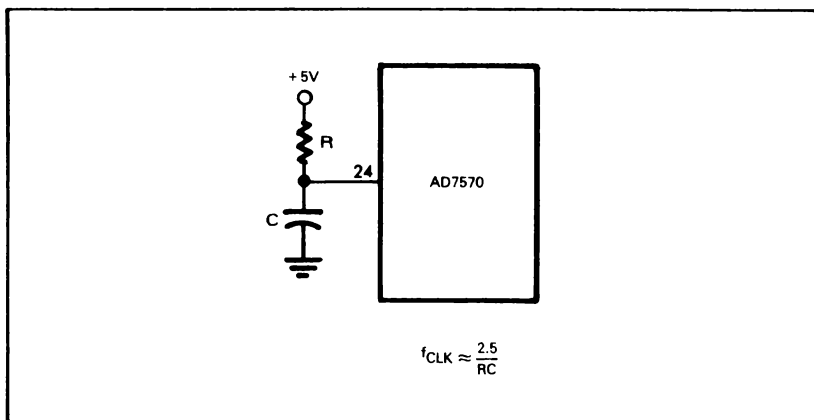
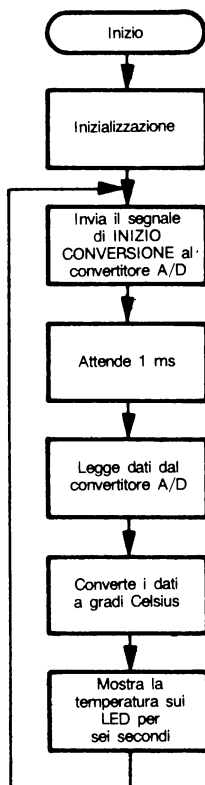



Figura 16-6
Generazione Di Una Frequenza Interna Di Clock

Diagramma di flusso generale del programma:

Descrizione del Programma:

1) Inizializzazione
La locazione 0 (locazione RESET del microprocessore 8080) contiene un salto al programma principale. Viene inizializzato soltanto il Puntatore dello Stack all'indirizzo RAM più elevato. Lo Stack è anche impiegato per immagazzinare gli indirizzi di ritorno di Subroutine.

2) Invio del segnale di INIZIO CONVERSIONE al convertitore A/D.
La CPU pulsa la linea di INIZIO CONVERSIONE prima posizionando un '1' sulla linea di uscita 7 e poi uno '0'.
Ogni ingresso dal convertitore richiede l'impulso di partenza.

3) Attende 1 ms per la conversione.
Un ritardo di 1 ms dopo l'impulso d'INIZIO CONVERSIONE garantisce una conversione completa. Attualmente il convertitore impiega al massimo 100 microsecondi per una conversione ad 8 bit. Si potrebbe ridurre il ritardo controllando il segnale BUSY dal convertitore. Questo segnale indicherà un '1' (conversione completa) oppure uno '0' (conversione in elaborazione) se la linea di ABILITAZIONE BUSY è indirizzata con un 1 logico. Nel caso attuale non c'è ragione per accelerare il processo di conversione.

4) Lettura dati dal convertitore A/D.
Una singola operazione d'ingresso legge i dati. Si potrebbe notare che l'AD7570J Analog Devices è dotato di un ingresso di ABILITAZIONE e di uscite tri-state cosicchè esso potrebbe essere collegato direttamente al Bus Dati del microprocessore.

5) Conversione dei dati in gradi Celsius.

IMPIEGO DI UNA TABELLA DI CALIBRAZIONE

La conversione impiega una tabella che contiene il valore dell'ingresso maggiore corrispondente ad una data temperatura. Il programma analizza la tabella cercando il valore maggiore o uguale al valore ricevuto dal convertitore. Il primo di tali valori trovati corrisponde alla temperatura richiesta; cioè se il decimo ingresso è il primo valore maggiore o uguale ai dati, la temperatura è 10 gradi. Questo metodo di ricerca è inefficiente ma l'applicazione attuale non ne richiede uno migliore. (Si veda Knuth, D.E., The Art of Computer Programming, Vol. 3 Sorting and Searching, Addison-Wesley, Reading, Mass, 1973.) Si noti che si deve mantenere il numero di ingresso in decimale piuttosto che binario. La sequenza di istruzione «ADI 1, DAA» conserva l'indice in due cifre decimali invece di un numero binario. Per esempio il numero d'ingresso dopo 9 (0001001 in binario) sarà il decimale 10 (00010000 binario) piuttosto che il binario dieci (00001010). La ragione di questa procedura è che si pianifica per mostrare la temperatura con due cifre decimali altrimenti si dovrebbe convertirla da binario a decimale.
La tabella potrebbe essere ottenuta mediante calibrazione o mediante approssimazione matematica. Il metodo di calibrazione è il più semplice, poichè il termometro deve essere calibrato comunque. La tabella occupa una locazione di memoria per ogni valore di temperatura.

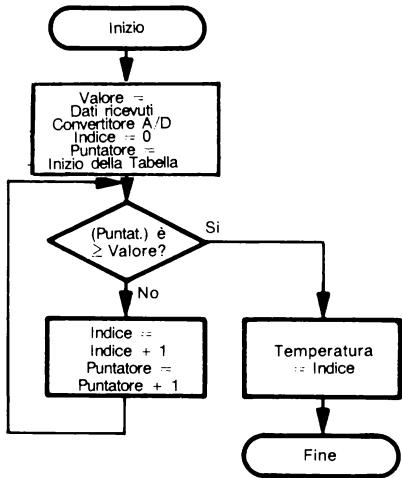
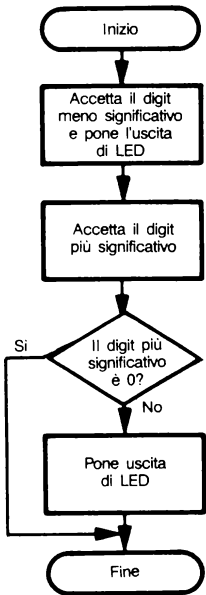
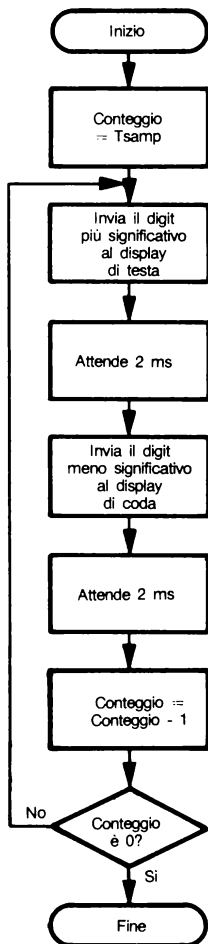
Diagramma di Flusso:

6) Prepara i dati per il display.

Diagramma di flusso:

La cifra meno significativa viene rimossa mediante una mascheratura e ponendo ad uno i bit che accendono il display con una istruzione di OR logico. Il risultato è conservato nel Registro E.

BLANKING DELLO ZERO NON SIGNIFICATIVO

La cifra più significativa si differenzia soltanto per il fatto che non si deve mostrare uno zero in testa (cioè i display mostrano «blank 7» piuttosto che «07» per 7 C). Questo comporta semplicemente la posizione a '0' del bit che commuta on i display se la cifra è zero. Il risultato è conservato nel registro D.

7) Mostra la temperatura per sei secondi.

Diagramma di flusso:

Ogni display viene pulsato abbastanza spesso cosicchè esso sembra essere illuminato con continuità. Se TPULS fosse reso più lungo (diciamo 50 ms) i display potrebbero apparire lampeggiare accesi e spenti.

Il programma impiega un contatore a 16 bit per contare il tempo tra i campionamenti di temperatura. L'8080 ha un'istruzione speciale. DCX o Doppio Decremento – per sottrarre '1' dal contatore a 16 bit. Comunque non esiste alcun modo per determinare direttamente quando il contatore raggiunge lo zero, poichè DCX non influenza i flag di stato. Così si può eseguire questa determinazione tramite l'OR logico degli otto bit più significativi e degli otto meno significativi del contatore. Se questo risultato è zero il contatore a 16 bit è zero.

```
;
;NOME DEL PROGRAMMA: TERMOMETRO
;DATA DEL PROGRAMMA: 7/22/76
;PROGRAMMATORE: LANCE A. LEVENTHAL
;REQUISITI DEL PROGRAMMA: 159 PAROLE
;REQUISITI RAM: NESSUNO
;REQUISITI I/O: 1 PORTA D'INGRESSO, 1 PORTA D'USCITA
;
;QUESTO PROGRAMMA È UN TERMOMETRO DIGITALE CHE ACCETTA L'INGRESSO
; DA UN CONVERTITORE A/D CONNESSO AD UN TERMISTORE, CONVERTE
; L'INGRESSO IN GRADI CELSIUS E MOSTRA I RISULTATI SU
; DUE DISPLAY LED A 7-SEGMENTI
;
;CONVERTITORE A/D
;
;IL CONVERTITORE A/D È UN 7570 ANALOG DEVICE CONVERTITORE
; MONOLITICO CHE FORNISCE UN'USCITA AD 8 BIT
;IL PROCESSO DI CONVERSIONE È AVVIATO DA UN IMPULSO SULLA LINEA
; DI INIZIO CONVERSIONE (BIT DI USCITA 7 DELLA PORTA ADOUT)
;LA CONVERSIONE È COMPLETATA IN 40 MICROSECONDI ED I DATI
; DIGITALI SONO DI TIPO LATCH
;
;
;DISPLAY
;
;SONO IMPIEGATI DUE DISPLAY LED A 7-SEGMENTI CON DECODIFICATORI
; SEPARATI (7447 OPPURE 7448 IN DIPENDENZA DEL TIPO DI DISPLAY)
;I DECODIFICATORI DEI DATI D'INGRESSO SONO CONNESSI AI BIT
; DA 0 A 3 DEL BUS DI USCITA DEL PROCESSORE
;IL BIT 4 DEL BUS DI USCITA DEL PROCESSORE È IMPIEGATO PER
; ATTIVARE I DISPLAY LED (IL BIT 4 È 1 PER INVIARE DATI AI LED)
;IL BIT 5 DEL BUS DI USCITA DEL PROCESSORE È IMPIEGATO PER
; SELEZIONARE QUALE LED DEVE ESSERE IMPIEGATO (BIT 5 A 1 SE
; DEVE ESSERE IMPIEGATO IL DISPLAY DI TESTA '0' SE DEVE ESSERE
; IMPIEGATO QUELLO DI CODA)
;
;METODO
;
;FASE 1 – INIZIALIZZAZIONE
; VIENE INIZIALIZZATO LO STACK DI MEMORIA (IMPIEGATO PER
; GLI INDIRIZZI DI RITORNO DI SUBROUTINE)
;FASE 2 – IMPULSO SULLA LINEA DI INIZIO CONVERSIONE
; LA LINEA DI INIZIO CONVERSIONE DEL CONVERTITORE A/D
; (BIT 7 DELLA PORTA ADOUT) VIENE PULSATA
;FASE 3 – ATTESA PER STABILIRE UN'USCITA A/D
; VIENE INTRODOTTA UN'ATTESA DI 1 MS PER PERMETTERE
; IL COMPLETAMENTO DELLA CONVERSIONE
```

```
;FASE 4 — LETTURA DI UN VALORE A/D E CONVERSIONE IN GRADI C
; VIENE IMPIEGATA UNA TABELLA DI CONVERSIONE CHE CONTIENE IL
; MASSIMO VALORE DELL'INGRESSO PER OGNI LETTURA DI TEMPERATURA
;FASE 5 — MOSTRA LA TEMPERATURA SUI LED
; LA TEMPERATURA VIENE MOSTRATA SUL LED PER SEI SECONDI
; PRIMA CHE VENGA ESEGUITA UN'ALTRA CONVERSIONE
;
;
;
;DEFINIZIONE DELLE VARIABILI DEL TERMOMETRO
;
;COSTANTI DEL SISTEMA DI MEMORIA
;
BEGIN    EQU      50H        ;BEGIN È LA LOCAZIONE INIZIALE DEL PROGRAMMA
LASTM    EQU      1000H      ;LASTM È L'INDIRIZZO INIZIALE PER LO STACK
;UNITÀ   I/O
;
ADIN     EQU 3               ;UNITÀ D'INGRESSO PER I DATI
                             ;DAL CONVERTITORE A/D
ADOUT    EQU 3               ;UNITÀ D'USCITA PER L'IMPULSO D'INIZIO
                             ;CONVERSIONE AL CONVERTITORE A/D
LDOUT    EQU                 ;LINEA DI USCITA PER I DISPLAY LED
;
;DEFINIZIONI
;
LEDON    EQU      00010000B  ;CODICE PER INVIARE L'USCITA AI LED
LEDSL    EQU      00100000B  ;CODICE PER SELEZIONARE IL DISPLAY
                             ;DI TESTA
MSCNT    EQU      131        ;CONTEGGIO IMPIEGATO NELLA ROUTINE DI
                             ;RITARDO PER IL RITARDO DI 1 MS
STCON    EQU      10000000B  ;USCITA CHE PONE AL LIVELLO ALTO
                             ;L'INIZIO CONVERSIONE
STTIM    EQU      1          ;RITARDO IN MS PER PERMETTERE ALL'A/D
                             ;DI ESEGUIRE LA CONVERSIONE
TPULS    EQU      2          ;LUNGHEZZA DELL'IMPULSO DEL DISPLAY IN MS
TSAMP    EQU      1500       ;TSAMP È IL NUMERO DI VOLTE CHE I DISPLAY
                             ;SONO PULSATI IN UN PERIODO DI CAMPIONAMENTO
                             ;DI TEMPERATURA, LA LUNGHEZZA DEL PERIODO
                             ;DI CAMPIONAMENTO È COSÌ 2* TPULS* TSAMP
                             ;MILLISECONDI, IL FATTORE 2* TPULS È INTRODOTTO DAL FATTO
                             ;CHE IL DISPLAY È PULSATO 2* TPULS VOLTE AL MS
;
;
ORG 0
;
;RIPRISTINA LA ROUTINE PER RAGGIUNGERE IL PROGRAMMA TERMOMETRO
         JMP      BEGIN      ;TROVA IL PROGRAMMA TERMOMETRO
;
;INIZIALIZZAZIONE DEL PROGRAMMA TERMOMETRO
;
         ORG      BEGIN
         LXI      SP,LASTM   ;PONE LO STACK ALLA FINE DELLA MEMORIA
;
;PULSA LA LINEA D'INIZIO CONVERSIONE PER FARE ESEGUIRE LA CONVERSIONE AD A/D
;
```

```
START   MVI     A,STCON
        OUT     ADOUT       ;INIZIO CONVERSIONE ALTO
        SUB     A
        OUT     ADOUT       ;INIZIO CONVERSIONE BASSO
;
;ATTENDE 1 MS PER LA CONVERSIONE
;
        MVI     A,STTIM     ;TEMPO DI RITARDO CONVERSIONE IN MS
        CALL    DELAY       ;ATTENDE LA CONVERSIONE
;
;LETTURA DATI DIGITALI DAL CONVERTITORE
;
        IN      ADIN        ;ACCETTA DATI DA A/D
;
;CONVERTE I DATI A/D A 2 DIGIT BCD
;
        CALL    CONVB       ;CONVERTE I DATI A BCD
;
;ACCETTA IL DIGIT MENO SIGNIFICATIVO
;
        MOV     B,A         ;CONSERVA I DIGIT BCD
        ANI     0FH         ;MASCHERA GLI LSD
        ORI     LEDON       ;PONE L'USCITA AI LED
        MOV     E,A         ;CONSERVA GLI LSD NEL REGISTRO E
;
;ACCETTA IL DIGIT PIÙ SIGNIFICATIVO BLANKING DELLO ZERO NON SIGNIFICATIVO
;
        MOV     A,B         ;RI-IMMAGAZZINA I DIGIT BCD
        RRC                 ;SPOSTA GLI MSD ALLA POSIZIONE
                            ;MENO SIGNIFICATIVA
        RRC
        RRC
        RRC
        ANI     0FH         ;MASCHERA MSD
        JZ      SVMSD       ;NON COMMUTARE IL DISPLAY DI TESTA SE MSD È ZERO
        ORI     LEDON       ;PONI L'USCITA AI LED
        ORI     LEDSL       ;SELEZIONA IL DISPLAY DI TESTA
SVMSD:  MOV     D,A         ;CONSERVA MSD NEL REGISTRO D
;
```

```
;PULSA I DISPLAY LED
;       LXI
                H,TSAMP     ;CONTATORE A 16 BIT PER NUMERO
                            ;DI IMPULSI DEL DISPLAY
DSPLY   MOV     A,D         ;PONE IL DIGIT DI TESTA
        OUT     LDOUT       ;USCITA AL DISPLAY DI TESTA
        MVI     A,TPULS     ;LUNGHEZZA DELL'IMPULSO DEL RITARDO DISPLAY
        CALL    DALAY
        MOV     A,E         ;ACCETTA IL DIGIT DI CODA
        OUT     LDOUT       ;USCITA AL DISPLAY DI CODA
        MVI     ATPULS      ;LUNGHEZZA DELL'IMPULSO DEL RITARDO DISPLAY
        CALL    DELAY
        DCZ     H           ;CONTO ALLA ROVESCIA DEL CONTATORE A 16 BIT
        MOV     A,H         ;TUTTI I SEGMENTI DEL CONTATORE AD 8 BIT
                            ;SONO A ZERO?
        ORA     L           ;RICORDA CHE L'ISTRUZIONE DCX NON PONE A ZERO
        JNZ     DSPLY       ;NO, CONTINUA A PULSARE I DISPLAY
        JMP     START       ;SÌ, ESEGUI ANCORA UN CAMPIONAMENTO DI TEMPERATUI
;
;LA SUBROUTINE DELAY ATTENDE IL NUMERO DI MILLISECONDI
;SPECIFICATO NEL REGISTRO A MEDIANTE CONTEGGIO COL REGISTRO C
;
;REGISTRI IMPIEGATI: A,C
;
DELAY:  MVI     C,MSCNT     ;CARICA IL REGISTRO C PER UN RITARDO DI 1 MS
WTLP:   DCR     C           ;ATTENDI 1 MS
        JNZ     WTLP
        DCR     A           ;CONTO ALLA ROVESCIA DEL NUMERO DI MS
        JNZ     DELAY
        RET
;
;LA SUBROUTINE CONVR CONVERTE L'INGRESSO DA UN CONVERTITORE
; A/D IN GRADI CELSIUS IMPIEGANDO UNA TABELLA,
; I DATI D'INGRESSO SONO NELL'ACCUMULATORE
; ED IL RISULTATO CONSISTE DI 2 DIGIT BCD POSIZIONATI NELL'ACCUMULATORE
;
;REGISTRI IMPIEGATI: A, B, C, H, L
;
CONVR:  LXI     H,DEGTB     ;ACCETTA L'INDIRIZZO DELLA BASE DELLA
                            ;TABELLA DI CONVERSIONE
        MOV     B,A         ;CONSERVA L'INGRESSO A/D
        MVI     C,O         ;INIZIA I GRADI A ZERO
CHVAL:  MOV     A,M         ;ACCETTA L'INGRESSO DALLA TABELLA
        CMP     B           ;L'INGRESSO A/D È MINORE O UGUALE ALL'INGRESSO?
        MOV     A,C         ;ACCETTA GRADI CELSIUS
        JNC     FOUND       ;SÌ, VALORE CORRETTO TROVATO
        ADI     1           ;SOMMA 1 A GRADI
        DAA                 ;ESEGUE BCD DI GRADI
        MOV     C,A
        INX     A           ;INGRESSO SUCCESSIVO ALLA TABELLA
        JMP     CHVAL
FOUND   RET                 ;RITORNO CON TEMP ESPRESSA CON DUE DIGIT IN A
;
;
;LA TABELLA DEGTB È STATA OTTENUTA DALLA CALIBRAZIONE
; CON UN RIFERIMENTO NOTO
```

```
; DEGTB CONTIENE IL VALORE MAGGIORE DELL'INGRESSO
; CHE CORRISPONDE AD UNA PARTICOLARE LETTURA DI TEMPERATURA
; (CIOÈ IL PRIMO INGRESSO È IL DECIMALE 58 COSÌ UN VALORE
; D'INGRESSO 58 È IL VALORE MAGGIORE CHE FORNISCE LA
; TEMPERATURA ZERO
; LETTURA — I VALORI SOTTO ZERO NON SONO CONSENTITI)
;
DEGTB   DB      58, 61, 63, 66, 69, 71, 74, 77, 80, 84, 87, 90, 93, 97, 101, 104, 108
        DB      112, 116, 120, 124, 128, 132, 136, 141, 145, 149, 154, 158, 163, 167
        DB      172, 177, 181, 186, 191, 195, 200, 204, 209, 214, 218, 223, 227, 232
        DB      236, 241, 245, 249, 253, 255
        END
```

Cod. 323 P

LANCE A. LEVENTHAL è un consulente specializzato in microprocessori e microprogrammazione ed è anche docente presso l'Engineering and Technology Department al Grossmont College di San Diego, California. È redattore tecnico della Society for Computer Simulation e di «Microprocessors in Simulations» per la rivista Simulation. È anche docente americano sui microprocessori per la IEEE, autore di oltre trenta articoli sui microprocessori e collaboratore abituale a pubblicazioni come Simulation, Electronic Design e Kilobaud.
La precedente esperienza del Dr. Leventhal comprende la collaborazione con Linkabit Corporation, Intelcom Read Tech., Naval Electronics Laboratory Center ed Harry Diamond Laboratories.
Egli ricevette il grado B.A. dall'Università di Washington a St. Louis, Missouri ed M.S. e Ph.D. dall'Università di California a San Diego. È membro di SCS, ACM ed IEEE.

GRUPPO
EDITORIALE
JACKSON

Lance A. Leventhal

8080A/8085: PROGRAMMAZIONE IN LINGUAGGIO ASSEMBLY

33